UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

“Red de área local virtual distribuida”

T E S I S

QUE PARA OBTENER EL GRADO DE

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

JONATHAN RICARDO BADILLO MEJÍA

DIRECTOR DE TESIS

DR. JOSÉ DAVID FLORES PEÑALOZA

CIUDAD UNIVERSITARIA, D.F.
2015

Jurado

Mat. Salvador López Mendoza
Mat. José Luis Torres Rodríguez
Dr. José David Flores Peñaloza
Dr. José de Jesús Galaviz Casas
M. en C. Fernanda Sánchez Puig

Agradecimientos

M*uchas gracias a*
U*stedes,*
C*ompañeros de vida. No*
H*abría*
A*lcanzado este nuevo gran logro,*
S*in todo su apoyo que siempre me han brindado.*

G*racias nuevamente,*
R*ealmente aprecio todo lo que han*
A*portado para poder lograr mi*
C*reciente superación personal.*
I*ncitándome a seguir adelante, siempre*
A*poyándome en las buenas y en las malas,*
S*iempre brindándome sabiduría para sobrellevar las situaciones más difíciles.*

A *todos ustedes les agradezco de corazón.*

T*odo este tiempo juntos, no lo cambiaría por ningún*
O*tro sin ustedes,*
D*aré gracias,*
O*tras mil veces*
S*i es necesario, para poder demostrarles mi gratitud.*

Variación de π-ema (Pi-ema) contando palabras por verso.

Dedicado a todos los que han tenido un gran impacto en mi vida.

A la Universidad Nacional Autónoma de México. Por darme la oportunidad para crecer intelectualmente y permitirme ser parte de ella.

A la H. Facultad de Ciencias. Por ser como mi segundo hogar durante mis estudios de licenciatura.

Al Dr José David Flores Peñaloza por su paciencia, su apoyo y sus consejos para realizar esta tesis y otros proyectos.

A todos mis profesores por su inmensa sabiduría que me han compartido.

A Mauricio, Elsa, Félix y Karen. Verdaderos amigos de confianza que me apoyaron en momentos de necesidad, me aconsejaron en momentos de incertidumbre y siempre estuvieron conmigo para pasar un buen rato.

A mis compañeros de taller de violín. Por compartir conmigo algo tan bello como es la música, permitirme vivir experiencias que no pudiera haber imaginado experimentar.

A Dios por permitirme vivir esta vida a lado de todos mis seres queridos.

Pero sobre todo, dedicado a mi madre, que siempre ha estado conmigo, en las buenas y en las malas, apoyándome para seguir adelante. Muchas gracias por todo mamá.

Índice de contenidos

1. Prefacio

"Nuestro vínculo en común más básico
es que todos habitamos este planeta..."
John F. Kennedy

1.1 Contenido de la tesis

Dentro de esta tesis se explicará qué es y cuáles fueron los motivos por los que se pensó en desarrollar al sistema *Atún*. Se dará una pequeña introducción sobre los conceptos principales que se utilizaron, tales como IPv4, UDP, VPN y TUN/TAP entre otros. También se detallará a fondo el funcionamiento del sistema y como se compone. Por último se plantearán las conclusiones a las que se llegaron con la realización de este sistema y se presentarán algunas propuestas para trabajos futuros.

1.2 Panorama general

Actualmente existen todo tipo de aplicaciones que funcionan por medio de Internet, como aplicaciones de mensajería instantánea, correo electrónico, videojuegos, sistemas de intercambio de archivos, etc. Algunas de las cuales suelen contar con mecanismos para funcionar en una LAN[1], proporcionando una mayor privacidad, mejores tiempos de respuesta, etc. Asimismo, existen otras aplicaciones creadas específicamente para ser utilizadas en este tipo de red[2]. Por otra parte, las relaciones interpersonales de cada persona, tanto familiares como sociales o laborales, en general se encuentran dispersas geográficamente, por lo que no pueden utilizar estas aplicaciones por no encontrarse en una misma LAN.

Existe un sistema que ya proporciona la posibilidad de utilizar este tipo de aplicaciones sin importar la ubicación geográfica de los integrantes de la red, llamado Hamachi. Es capaz de crear redes privadas virtuales entre equipos remotos de forma segura y cuenta con su propio servidor para gestionar las redes; sin embargo, todos los usuarios deben contar con una suscripción de pago o gratuita para poder crear o unirse a una red. Las suscripciones de pago permiten contar con hasta 256 miembros en una red, mientras que la suscripción gratuita solo permite hasta 5 miembros[3].

1.3 Objetivo

Se pretende crear un nuevo sistema que igualmente sea capaz de crear redes privadas virtuales pero que le dé la libertad a las personas de ubicar al servidor donde ellos prefieran, que puedan tener una mayor cantidad de usuarios dentro de una misma red y que puedan estar conectados a varias redes simultáneamente, todo bajo una *Licencia Pública General GNU GPL*.

2. Introducción

"El principio es la parte
más importante del trabajo"
Platón

Atún es una implementación de una LAN distribuida sobre IP, la cual le permite a un grupo de equipos comunicarse directamente a nivel de capa de red aún cuando no se encuentren en una misma ubicación geográfica.

Cuenta con una arquitectura Cliente – Servidor cuyos mensajes de control se envían vía servicios XML – RPC y cuyos mensajes de datos se envían sobre UDP.

Los clientes proveen a sus kernels de interfaces de red

virtuales, utilizando el controlador TUN/TAP, las cuales se establecen en las LAN's distribuidas. De esta forma, los procesos que se ejecutan sobre esos kernels pueden tener acceso a la LAN distribuida sin que sea necesaria ninguna modificación sobre ellos.

El servidor se encarga de administrar las LAN's distribuidas y de reenviar los mensajes de datos entre los diferentes clientes pertenecientes a una misma LAN distribuida.

Atún está diseñado e implementado de tal forma que los clientes pueden comunicarse con el servidor aun cuando se encuentren detrás de NAT's.

En este capítulo se establecerán todos los conceptos preliminares sobre las tecnologías que sustentan a *Atún*, incluyendo las aquí mencionadas.

2.1 Protocolo IPv4

El protocolo de Internet es un protocolo bien conocido el cual proporciona bloques de datos llamados datagramas, para ser transmitidos en redes de computadoras de intercambio de paquetes. Los dispositivos se identifican por medio de direcciones con un tamaño fijo ubicadas en el encabezado del datagrama. Además, también ofrece, de ser necesario, fragmentación y reensamblado de grandes datagramas para ser transmitidos a través de redes que solo pueden mandar pequeños paquetes de datos[4].

La versión mayormente utilizada del Protocolo de Internet es IPv4[5] el cual cuenta con un encabezado con 14 campos para describir al datagrama y asegurar su integridad, aunque no su

recepción. Las direcciones que representan a los dispositivos origen y destino son campos de 32 bits, dando lugar a un total de 4,294,967,296 direcciones posibles.

Figura 1: Encabezado de un datagrama IPv4

Algunos campos destacables del encabezado de un datagrama IPv4 son:

- *Versión*. La versión del protocolo IP. En este caso es 4.
- *IHL*. Internet Header Length. La longitud del encabezado en palabras de 4 bytes, el valor mínimo es 5, que significa sin opciones.
- *Longitud Total*. La longitud total del datagrama, incluyendo el encabezado, medido en bytes. Al ser un campo de 16 bits, el tamaño máximo de un datagrama IP es de 65,535 bytes.
- *TTL*. Time To Live. Previene que un datagrama viaje en círculos por siempre, limitando su tiempo de vida. Cuando el datagrama llega a un ruteador, éste le disminuye su tiempo de vida, y si llega a 0 entonces el datagrama es descartado.
- *Protocolo*. Define el protocolo utilizado dentro de la información del datagrama IP.
- *Checksum*. Suma de verificación utilizada para la detección de errores del encabezado.
- *Dirección IP Origen*. Indica la dirección IP del emisor del

datagrama.

- *Dirección IP Destino*. Indica la dirección IP del receptor del datagrama.

Existe otra versión utilizada actualmente llamada IPv6 planeada para ser la sucesora de IPv4. La principal motivación para desarrollarla fue que el espacio de direcciones disponibles de 32 bits de IPv4 comenzaba a agotarse. IPv6 cuenta con un espacio de direcciones de 128 bits, dando lugar a mas de 340 sextillones de direcciones posibles. El diseño de IPv6 también dio la oportunidad de ajustar y mejorar otros aspectos de IPv4[5].

Sin embargo, aun cuando el 6 de junio del 2012 se lanzó una iniciativa para implementar de forma permanente IPv6 para los productos y servicios de los participantes y éstos ya han realizado dicha implementación[6], al parecer la mayoría de los usuarios aun no utilizan esta versión. En la figura 2 se muestra que solo el 3.58% de los usuarios acceden a Google a través de IPv6[7].

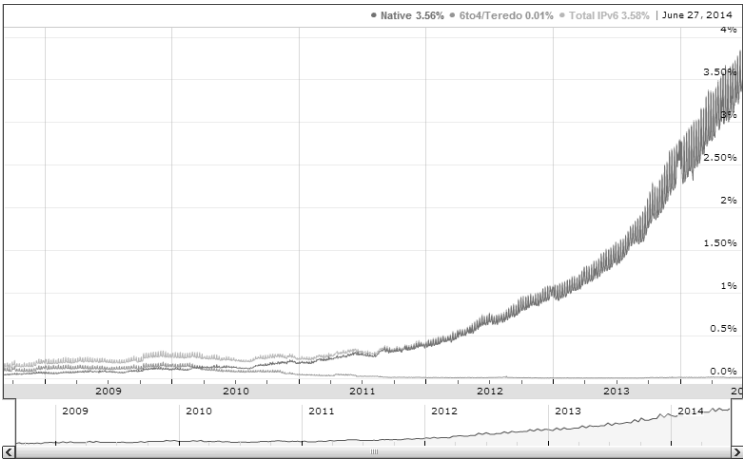

Figura 2: Adopción de IPv6 de usuarios de Google

2.2 Protocolo UDP

TCP provee una transferencia de información confiable. Usando diferentes técnicas, TCP se asegura de que los segmentos sean enviados y recibidos de forma correcta y en orden. También provee un control del congestionamiento aunque esto es más como un servicio para la Internet como un todo que uno para las aplicaciones que recurren a él. Pero todas estas técnicas le añaden una sobrecarga a cada segmento, además de un retraso por tener que establecer una conexión antes de comenzar a enviar información[5].

En ocasiones, como en las aplicaciones en tiempo real, es necesario que los segmentos se envíen rápidamente y se puede tolerar la pérdida de algunos de ellos, como en aplicaciones de

transmisión continua como videollamadas. En estos casos es cuando el protocolo UDP puede funcionar mejor.

UDP no le agrega más cosas al Protocolo de Internet que un pequeño encabezado de tan solo 8 bytes, generando una menor sobrecarga. Luego pasa el segmento resultante pasa directamente a la capa de red, proporcionando un control más sutil a nivel de capa de aplicación sobre lo que se envía, y cuando se envía. También, dado que no existe algo como el *handshaking* de TCP, se dice que UDP es sin conexión, quitándole ese retraso extra que se utiliza para establecer una conexión[5].

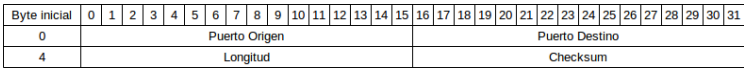

Figura 3: Encabezado de un segmento UDP

Los campos del encabezado de un segmento UDP son:

- *Puerto Origen*. Puerto del equipo emisor por el cual es enviado el segmento.
- *Puerto Destino*. Puerto del equipo receptor por el cual es recibido el segmento.
- *Longitud*. Longitud total del segmento, incluyendo el encabezado, medido en bytes. Dado que la longitud es un campo de 16 bits, la longitud máxima del segmento es de 65,535 bytes.
- *Checksum*. Suma de verificación utilizada para la detección de errores en el segmento.

2.3 VPN

Una red privada virtual es una extensión de una red privada que abarca enlaces a través de redes compartidas o públicas como la Internet, permitiendo enviar información a través de tales redes de tal manera que se emulan las propiedades de un enlace privado punto a punto.

Para emular un enlace punto a punto, la información es encapsulada con un encabezado que proporciona información de ruteo, permitiéndole alcanzar su equipo destino, a este paso de la conexión se le conoce como túnel. Para emular un enlace privado, la información es cifrada para su confidencialidad, de tal manera que sea indescifrable si es interceptada en la red, a este paso de la conexión se le conoce como la conexión a la red privada virtual.

Entre los usos más comunes de una VPN se encuentran los accesos remotos por medio de Internet, que permiten tener acceso a recursos remotos manteniendo la privacidad de éstos; conexiones de diferentes redes por medio de Internet y conectar computadoras por medio de una Intranet para permitir accesos a redes seguras u ocultas[8].

2.4 Tunneling

El *Tunneling* es un método que utiliza una red auxiliar como medio de transporte para enviar datos entre otras dos redes que normalmente no pudieran comunicarse entre si. Estos datos pueden ser marcos o paquetes de otros protocolos y en lugar de ser enviados directamente como se les creó. El proceso del *Tunneling* primero captura la información que se pretende enviar y la encapsula con un encabezado adicional, el cual proporciona información de ruteo. Después se transmite el nuevo paquete por la red auxiliar y al llegar a su destino se desencapsula y se reenvia la información original a su destinatario que ya es alcanzable desde su propia red.

Las tecnologías de tunneling pueden basarse en protocolos de capa de enlace, que utilizan marcos como su unidad de intercambio, o de capa de red, que utilizan paquetes. Ejemplos de protocolos de tunneling: *Point to Point Tunneling Protocol* (PPTP) y *Layer Two Tunneling Protocol* (L2TP) de la capa de enlace e *IP Security* (IPSec) de la capa de red.

Para las tecnologías de tunneling de capa de enlace un túnel es como una sesión, ambos extremos deben aceptar el túnel y negociar la configuración de las variables, tales como asignación de direcciones o parámetros de cifrado o compresión. Un protocolo de mantenimiento del túnel es utilizado como un mecanismo para administrar el túnel.

Las tecnologías de tunneling de capa de red generalmente suponen que todo lo relacionado con la configuración ha sido establecido previamente, a menudo por procesos manuales. Para estos protocolos no hay fase de mantenimiento del túnel.

Una vez que el túnel se ha establecido, se pueden enviar datos a través de él. Ambos extremos utilizan algún protocolo de transferencia de datos por medio de túnel para preparar los datos antes de ser enviados[8].

2.5 NAT

Las presiones combinadas que ejercen el gran crecimiento de Internet y sus retos masivos de seguridad la han forzado a evolucionar en formas que les dificulta la vida a muchas aplicaciones.

La arquitectura original de direcciones, en la que cada nodo poseía una dirección IP única y podía comunicarse directamente con cualquier otro nodo, ha sido reemplazada por una nueva arquitectura que consiste en una red principal de direcciones globales y muchas redes de direcciones privadas interconectadas por medio de NAT's.

En esta nueva arquitectura, solo los nodos de la red principal de direcciones globales pueden ser fácilmente contactadas desde cualquier lugar de la red, debido a que solo ellos poseen direcciones IP únicas, ruteables globalmente. Los nodos en las redes privadas pueden conectarse con otros nodos dentro de su misma red privada, y usualmente también pueden abrir conexiones TCP o UDP a nodos de la red principal de direcciones globales.

Aunque el espacio de direcciones de IPv6 eventualmente podrá reducir la necesidad de NAT's, a corto plazo, IPv6 está incrementando su demanda, ya que ellos por si mismos proporcionan la manera más fácil de lograr interoperabilidad entre los dominios de

direcciones de IPv4 e IPv6. Y el anonimato e inaccesibilidad a nodos en redes privadas ha sido ampliamente percibidos como beneficios de seguridad y privacidad[9].

Una sesión UDP o TCP de NAT se compone del tráfico que se administra como una unidad para la traducción. Todas las sesiones de un NAT se identifican de forma única con la tupla (IP Origen, Puerto TCP/UDP Origen, IP Destino, Puerto TCP/UDP Destino) y su dirección (entrante o saliente) depende del paquete que inicie la sesión.

El tipo básico de un NAT provee un puente asimétrico entre una red privada y una pública. Por defecto este tipo de NAT permite que solo las sesiones salientes puedan atravesarlo, los paquetes entrantes son descartados a menos que el NAT los identifique como parte de una sesión existente iniciada desde la red privada a menos que se le asigne una regla particular para que permita el paso libre hacia algún nodo particular. Para los paquetes salientes, la dirección IP Origen y sus demás campos relacionados como los checksums de IP, TCP y UDP son traducidos por el NAT. Para los paquetes entrantes, la dirección IP Destino y sus respectivos checksums como los de arriba son traducidos por el NAT.

Otro tipo de NAT extiende la noción de traducción a la siguiente capa, traduciendo los identificadores de la capa de transporte, es decir, los puertos de TCP o UDP. Ésto le permite a un conjunto de nodos compartir una misma dirección externa[10].

Cuando un extremo dentro de una red privada abre una sesión saliente a través de un NAT, éste le asigna una regla de filtrado para el mapeo entre las tuplas de dirección y puertos internos (X:x) y externos (Y:y).

Para esta regla de filtrado los NAT's pueden tener diferentes comportamientos para determinar como serán filtrados los paquetes

provenientes de un extremo externo específico. Estos pueden ser:

- Independiente del Extremo.

El NAT filtra solo los paquetes que no se encuentren destinados a un extremo interno (X:x) activo, independientemente del extremo externo (Z:z). El NAT reenvía cualquier paquete destinado a (X:x). En otras palabras, enviar paquetes desde el lado interno del NAT a cualquier dirección IP externa es suficiente para permitir cualquier paquete de vuelta al extremo interno desde cualquier extremo externo diferente.

- Dependiente de la dirección IP.

El NAT filtra los paquetes que no se encuentren destinados a un extremo interno (X:x). Además, el NAT filtra todos los paquetes de un extremo externo (Y:y) si (X:x) no le ha enviado paquetes con anterioridad a cualquiera de los puertos de Y (Y:cualquiera). En otras palabras, para recibir paquetes de un extremo externo en particular, es necesario que el extremo interno le haya enviado primero un paquete a su dirección IP.

- Dependiente de la dirección IP y puerto.

Similar al comportamiento anterior, salvo que ahora el puerto del extremo externo también es relevante. El NAT filtra los paquetes que no se encuentren destinados a un extremo interno (X:x). Además, el NAT filtra todos los paquetes de un extremo (Y:y) si (X:x) no le ha enviado paquetes con anterioridad exactamente a (Y:y). En otras palabras, para recibir paquetes de un extremo externo en particular, es necesario que el extremo interno le haya enviado primero un paquete a la dirección IP y puerto de tal extremo externo[11].

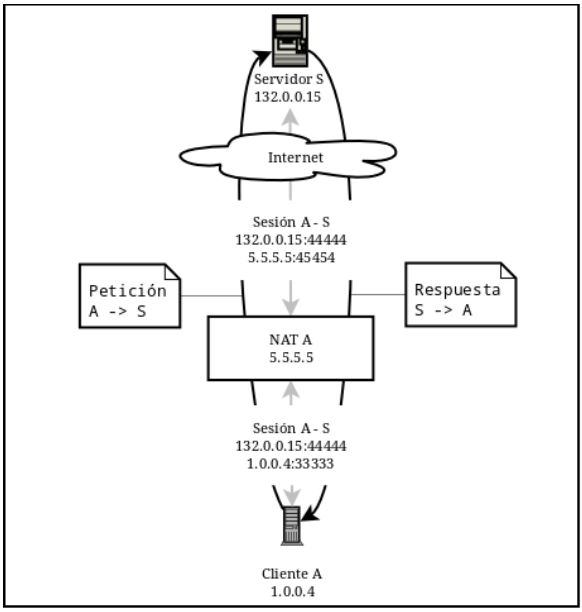

Figura 4: Funcionamiento de un NAT.

2.6 Relaying

Es un método seguro pero poco eficiente de comunicación P2P a través de NAT que hace ver a la comunicación como una de tipo Cliente – Servidor normal, por medio de redireccionamiento de paquetes.

Para esto, es necesario que los equipos se encuentren conectados, ya sea por TCP o UDP, a un servidor externo con una IP accesible por todos por medio de sesiones de NAT. Después, en lugar

de mandarse mensajes directamente, lo cual es imposible debido a que sus respectivos NAT's previenen la comunicación directa entre ellos, los mensajes se mandan al servidor y éste se los reenvía a sus respectivos destinatarios.

Sus desventajas son que al requerir estar los clientes conectados al servidor, consumen el poder de procesamiento y ancho de banda del servidor y pueden aumentar la latencia de la comunicación entre los clientes. Como consecuencia, ésta no es una solución escalable. Sin embargo, dado que no hay otra técnica más eficiente que funcione de manera segura en todos los tipos de NAT, es una buena estrategia si se desea la máxima robustez[9].

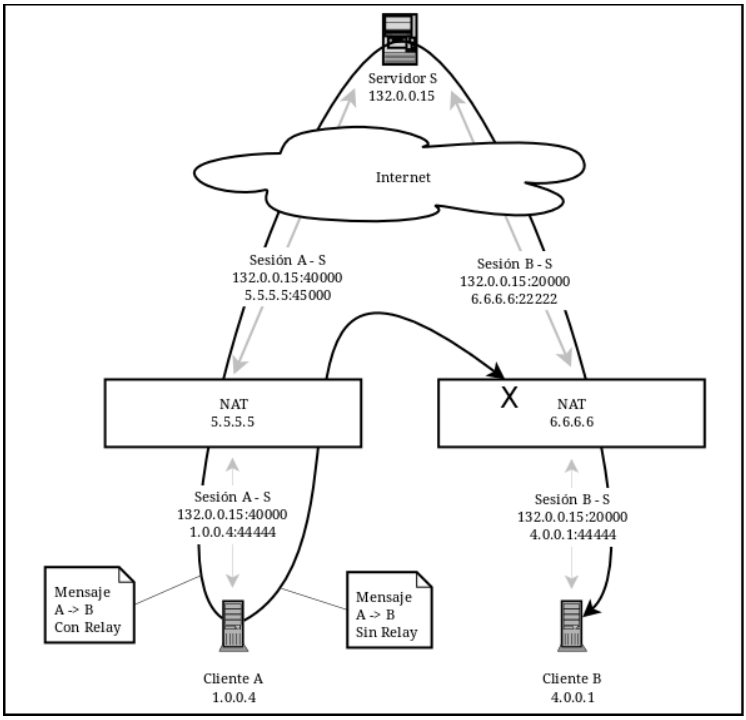

Figura 5: Funcionamiento de Relay

UDP Hole Punching

La nueva arquitectura de direcciones, mencionada anteriormente, es adecuada para la comunicación Cliente – Servidor típica, donde el cliente se encuentra en una red privada y el servidor en la red principal de direcciones. Sin embargo, dificulta que 2 nodos en redes privadas diferentes se comuniquen entre sí directamente, lo

cual es a menudo importante en los protocolos de comunicación peer-to-peer que se utilizan en aplicaciones de telecomunicaciones y juegos en línea. Por ello es necesario lograr que dichos protocolos funcionen sin problemas en presencia de NAT.

A uno de los más efectivos métodos para la comunicación peer-to-peer entre nodos en diferentes redes privadas se le conoce como *Hole Punching*. Esta técnica ya es ampliamente utilizada en aplicaciones basadas en el protocolo UDP, pero en esencia la misma técnica funciona también para TCP. Hole Punching no compromete la seguridad de una red privada, en lugar de eso, habilita a las aplicaciones para funcionar dentro de las políticas de seguridad predeterminadas de la mayoría de los NAT's, señalándoles efectivamente a todos los NAT's en el camino, que las sesiones de comunicación peer-to-peer fueron “solicitadas” y, por lo tanto, deben ser aceptadas.

UDP Hole Punching permite a dos clientes establecer una sesión UDP directa peer to peer con la ayuda de un servidor de encuentros conocido, aun si ambos se encuentran detrás de NAT.

Hole Punching supone que ambos clientes, A y B, ya cuentan con una sesión UDP activa con el servidor de encuentros S.

Cuando un cliente se registra con S, el servidor guarda 2 *extremos* para ese cliente, la pareja (dirección IP, puerto UDP) que el cliente utiliza dentro de su red privada (asignados por el NAT), y la pareja (dirección IP, puerto UDP) que el servidor observa del cliente al comunicarse con él (dirección y puerto del NAT). La primera pareja será referida más adelante como el *extremo privado* del cliente y la segunda como el *extremo público*. El cliente puede proporcionarle su extremo privado al servidor en un campo en el cuerpo del mensaje de registro, y el extremo público de los encabezados IP y UDP del mismo mensaje de registro. Si el cliente no está detrás de un NAT entonces ambos extremos son idénticos.

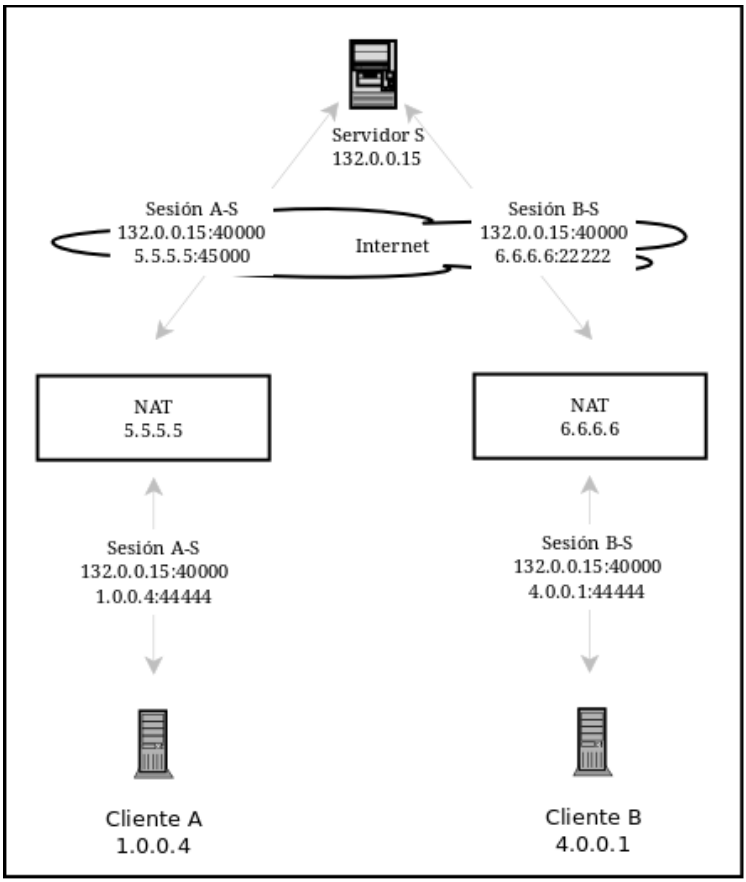

Figura 6: Estado antes de Hole Punching

Si un cliente A quiere establecer una sesión UDP directamente con un cliente B, Hole punching procede de la

siguiente manera:

1. Inicialmente, A no conoce como contactar a B, así que A le pide ayuda a un servidor S para establecer una sesión UDP con B.

2. S le responde a A con un mensaje que contiene los extremos público y privado de B. Al mismo tiempo, S usa su sesión UDP con B para enviarle un mensaje de petición de conexión que contiene tanto los extremos público y privado de A.

3. Cuando A recibe los extremos de B, A comienza a enviarle paquetes a ambos extremos y observa de cual de ellos obtiene una respuesta válida de B. De igual forma, cuando B recibe la petición de conexión con los extremos de A, B comienza a enviarle paquetes a ambos extremos buscando uno que funcione.

El envío de los paquetes al extremo privado de los clientes se debe a la posibilidad de que ambos clientes no sepan que se encuentren detrás de un mismo NAT por lo que se encontrarían en la misma red privada. Dado que ésta ruta suele ser más rápida que una ruta indirecta a través del NAT, los clientes suelen usar los extremos privados para las comunicaciones siguientes.

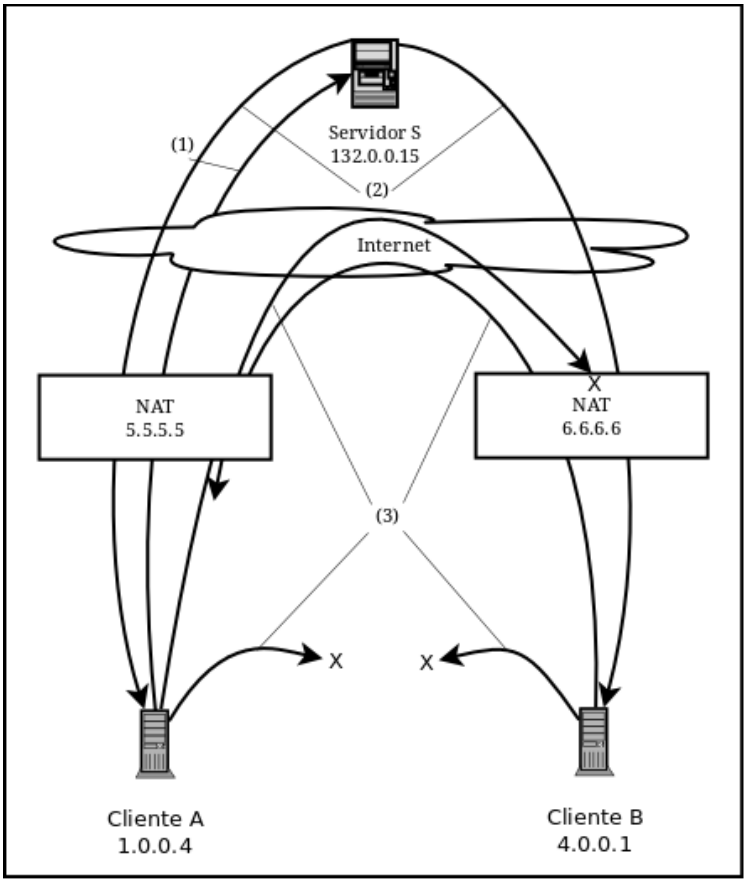

Figura 7: Mensajes durante el Hole Punching

El caso más común es en el que ambos clientes se encuentren en redes privadas detrás de NAT's diferentes. En este caso, los extremos privados no son ruteables globalmente y los mensajes enviados a estos extremos pueden llegar a un nodo equivocado o a ninguno. Ésto debido a que muchos NAT's también funcionan como

servidores DHCP proporcionando direcciones IP de un conjunto de direcciones usualmente determinado por el vendedor del NAT, así que es probable que el mensaje de A, enviado al extremo privado de B, llegue a un nodo incorrecto en la red privada de A que, casualmente, tenga la misma dirección IP privada que B. Así que las aplicaciones deben autenticar todos los mensajes de tal forma que se filtre todo el tráfico perdido. Los mensajes podrían tener nombres específicos de la aplicación, tokens de cifrado o aleatorios configurados previamente en el servidor.

Considerando el paso número 3 en este caso, el primer mensaje de A enviado al extremo público de B es notado por el NAT como el primer paquete UDP de una sesión saliente. El origen de la nueva sesión es el mismo que el utilizado en la sesión existente entre A y S, pero su destino es diferente. Si el NAT de A tiene un buen comportamiento, es decir, es independiente del extremo, entonces el NAT mantiene la identidad del extremo privado de A, traduciendo consistentemente todas las nuevas sesiones salientes del extremo privado de A a su extremo público utilizado con S en lugar de crear un nuevo extremo si las sesiones van dirigidas a algún otro destinatario. Así, el primer mensaje de A hacia B abre un agujero en el NAT de A para la nueva sesión UDP identificada con el extremo privado de A y el extremo público de B en la red privada de A, y con el extremo público de A y el extremo público de B en Internet.

Si el mensaje de A, enviado al extremo público de B llega al NAT de B antes de que el primer mensaje de B, enviado al extremo público de A, cruce su propio NAT, entonces el mensaje de A puede ser interpretado como tráfico entrante no solicitado y descartado por el NAT de B. Sin embargo, el primer mensaje que B envía hacia el extremo público de A abre de forma similar un agujero en el NAT de B para la nueva sesión UDP identificada con el extremo privado de B y el extremo público de A en la red privada de B, y con el extremo

público de B y el extremo público de A en Internet.

Una vez que los primeros mensajes de A y B crucen sus respectivos NAT's, habrá agujeros en ambas direcciones y la comunicación UDP puede proceder con normalidad. Cuando los clientes verifiquen que su extremo público funciona, pueden dejar de enviarse mensajes por su extremo privado.

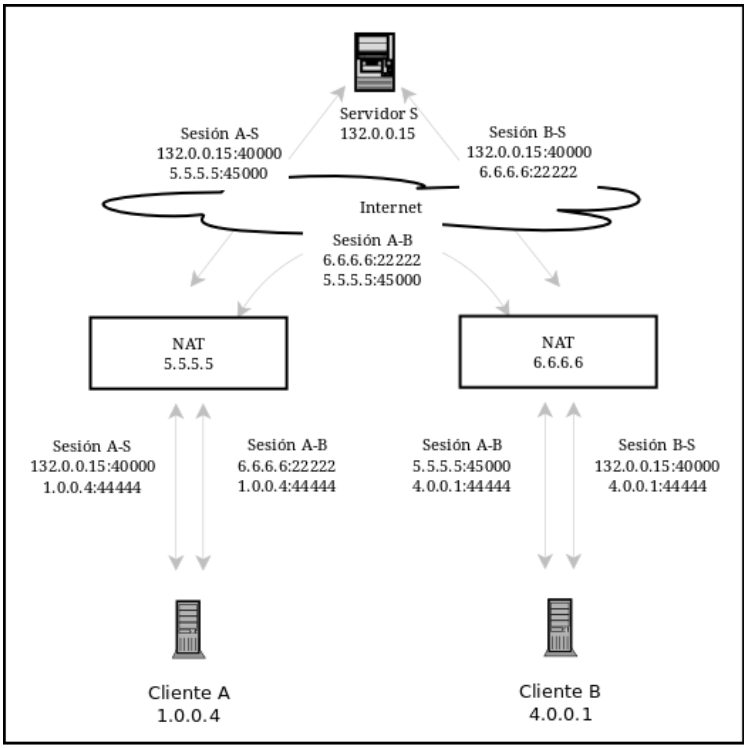

Figura 8: Estado después de Hole Punching

Dado que el protocolo UDP no provee al NAT con una forma confiable e independiente de la aplicación para determinar el tiempo de vida de una sesión que lo atraviese, la mayoría de los NAT simplemente asocian un temporizador de inactividad en sus tablas para las comunicaciones UDP, cerrando el agujero si no hay tráfico que lo utilice en cierto periodo de tiempo. Desafortunadamente no existe un valor estándar para el límite de tiempo de inactividad, por lo que algunos NAT's podrían tener límites de solo algunos segundos. Si una aplicación necesita mantener una sesión inactiva después de haberla establecido mediante *Hole Punching*, la aplicación debe enviar paquetes periódicamente para mantener con vida la sesión para asegurar que el estado de traducción relevante en el NAT no desaparezca. Muchos NAT suelen asociar estos temporizadores a una sola sesión UDP identificada por una pareja de extremos en particular, por lo que estos paquetes para mantener viva una sesión no funcionarían en otras, aun cuando todas las sesiones se originen en el mismo extremo privado, por lo que en lugar de enviar estos paquetes a cada una de las sesiones, las aplicaciones pueden detectar cuando una sesión UDP deja de funcionar y reutilizar el método de Hole Punching bajo demanda[9].

2.8 TUN/TAP

TUN/TAP es un controlador para dispositivos de red virtuales el cual proporciona recepción y transmisión de paquetes, para programas que se encuentran alojados en el espacio de usuario. Puede ser visto como un simple dispositivo Punto a Punto o de Ethernet, el cual en lugar de recibir paquetes desde un medio físico los recibe desde un programa en el espacio de usuario, y en lugar de enviar los paquetes por un medio físico los escribe en el programa en el espacio de usuario.

TUN es un dispositivo virtual de red Punto a Punto. Su

controlador está diseñado como un soporte de bajo nivel para el Kernel para *Tunneling* a nivel de capa de red y proporciona a la aplicación en el espacio de usuario el dispositivo /dev/tunX y la interfaz virtual tunX.

Las aplicaciones en el espacio de usuario pueden escribir datagramas IP en /dev/tunX y el Kernel los recibirá desde la interfaz tunX; y de igual manera, cada datagrama IP que el Kernel escriba en tunX pueden ser leídos por las aplicaciones desde el dispositivo /dev/tunX.

Un dispositivo TAP es un dispositivo virtual de red Ethernet su controlador está diseñado como soporte de bajo nivel para el Kernel para *Tunneling* a nivel de capa de enlace y proporciona al espacio de usuario el dispositivo /dev/tapX y la interfaz virtual Ethernet tapX.

Las aplicaciones en el espacio de usuario pueden escribir marcos Ethernet en /dev/tapX y el Kernel los recibirá en la interfaz tapX; y de igual manera, cada marco Ethernet que el Kernel escriba en tapX puede ser leído por las aplicaciones desde el dispositivo /dev/tapX.

Cuando una aplicación abre el controlador TUN/TAP, ésta crea y registra el dispositivo de red correspondiente tunX o tapX. Luego de que la aplicación cierre los dispositivos, el controlador automáticamente los eliminará junto con todas sus rutas correspondientes.

Al momento de crear el dispositivo, es necesario proporcionar una bandera para identificar su tipo y poder generar los encabezados correspondientes. Las banderas disponibles son:

IFF_TUN	Para un dispositivo TUN
IFF_TAP	Para un dispositivo TAP
IFF_NO_PI	Para no proporcionar información extra.

Si la bandera IFF_NO_PI no se establece, cada paquete se encapsula con un encabezado adicional de 4 bytes.

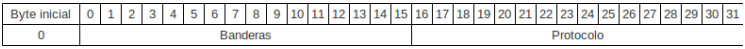

Figura 9: Encabezado de un paquete TUN/TAP.

Campos:

Banderas	Para identificar el tipo de dispositivo.
Protocolo	Protocolo del datagrama contenido en el paquete[12].

2.9 Arquitecturas de aplicaciones de red

Una arquitectura de aplicación es la que dicta cómo la aplicación será estructurada sobre varios sistemas finales. Las 2 arquitecturas predominantes son: Cliente – Servidor y Peer to Peer.

En una arquitectura Cliente – Servidor, siempre existe un equipo que se encuentra activo en todo momento, llamado el servidor, el cual sirve peticiones de muchos otros equipos, llamados clientes. Los clientes no necesariamente tienen que estar siempre activos. Otra característica de esta arquitectura es que el servidor tiene una dirección IP fija y bien conocida. Gracias a éstas característica del servidor, un cliente siempre puede contactarlo

enviando un paquete a su dirección.

Entre sus ventajas se encuentra la seguridad ya que las peticiones pueden ser monitorizadas y registradas, permitiendo que solo aquellos que tengan los permisos requeridos puedan tener acceso, previniendo los accesos no autorizados. Por otra parte, cuenta con una desventaja importante, a menudo para una aplicación que utiliza esta arquitectura, un solo servidor no es capaz de responder a todas las peticiones de sus clientes. Una solución que suele usarse para este problema es la utilización de un grupo de equipos para crear un servidor virtual más poderoso, aunque el mantenimiento de su infraestructura puede ser muy costoso.

En una arquitectura Peer-To-Peer, a veces referida solo como P2P, no existe como tal una dependencia a un servidor que se encuentre siempre activo. En lugar de ello, las aplicaciones se aprovechan de una comunicación directa entre parejas de equipos conectados intermitentemente llamados peers (iguales). Esto es por lo que se le llama Peer-To-Peer (de igual a igual). Los peers no le pertenecen al proveedor del servicio sino a cada uno de los usuarios.

Una de sus principales ventajas es su auto escalabilidad. Por ejemplo, para una aplicación P2P de intercambio de archivos, aunque cada peer le genera una carga de trabajo, también le añade capacidad de servicio al sistema distribuyendo a otros peers los archivos que ya haya descargado.

Sin embargo, las futuras aplicaciones P2P enfrentan 3 grandes riesgos:

Los ISP han sido creados pensando en un uso asimétrico del ancho de banda, dándole mucho mayor relevancia al tráfico de descarga que al de carga, pero las aplicaciones P2P invierten esa relevancia, creándoles un significante estrés a los ISP por lo que

estas aplicaciones deberían ser amigables con ellos.

Dada su naturaleza distribuida y abierta, asegurar este tipo de aplicaciones es un verdadero reto.

El éxito de estas aplicaciones también depende de convencer a los usuarios de que voluntariamente compartan su ancho de banda, almacenamiento y recursos computacionales entre ellos[5].

2.10 Protocolo XML-RPC

Las llamadas a procedimientos remotos, que llamaremos simplemente RPC, parecen ser un paradigma útil para brindar comunicación a través de una red entre programas escritos en un lenguaje de alto nivel.

La idea de las RPC se basa en la observación de que las llamadas a procedimientos son un mecanismo bien conocido y entendido para transferir el control y los datos dentro de un programa ejecutándose en una sola computadora, por lo que se propone que ese mismo mecanismo sea extendido para brindar el mismo tipo de transferencia a través de una red de comunicación.

Cuando un procedimiento remoto es invocado, el entorno que lo llama es suspendido, los parámetros se envían a través de la red al entorno en el que se procedimiento será ejecutado y el procedimiento deseado se ejecuta ahí. Cuando el procedimiento termina y produce sus resultados, éstos son enviados de regreso al entorno que lo llamó, donde la ejecución continua como si regresara desde una simple

llamada en una sola computadora.

Mientras el entorno que llama al procedimiento es suspendido, otros procesos en la computadora pueden seguir ejecutándose dependiendo de los detalles de paralelismo del entorno y implementación de las RPC[13].

XML-RPC es un protocolo diseñado para ser lo más simple posible que permite hacer RPC utilizando una petición de tipo POST de HTTP y cuyo cuerpo está construido en formato XML, el servidor procesa la petición y devuelve los resultados también en formato XML.

El formato de la URI en la primera línea del encabezado no está especificada. Podría estar vacía o ser una sola diagonal si el servidor solo administra peticiones XML-RPC o, si el servidor administra varios tipos de peticiones, la URI ayuda a dirigir la petición al código que la administra. Se deben especificar los encabezados de HTTP “Host”, “User-Agent”, “Content-Type = text/xml”, y “Content-Size” con el valor correcto.

La carga útil se encuentra en formato XML en una sola estructura <methodCall>. Esta estructura debe contener a lo más 2 estructuras internas, una <methodName> que contenga una cadena de texto con el identificador del procedimiento que será llamado y, de requerir parámetros, una estructura <params> con cualquier cantidad de <param> internos, cada uno con su respectivo valor <value>.

Los valores de los parámetros pueden ser estructuras simples como números (<int>, <double>, <boolean>), cadenas (<string>), fechas (<date>), cifrado en base 64 (<base64>); o estructuras más complejas (<struct>) y arreglos (<array>).

A menos que exista un error en un nivel inferior, la respuesta siempre será un 200 OK de HTTP. El cuerpo de la respuesta es una sola estructura XML <methodResponse>, la cual contiene un único valor de respuesta en un formato como el de los parámetros del procedimiento. Una respuesta también puede tener, pero no al mismo tiempo, una estructura <fault> con un valor de tipo <struct> con dos miembros: <faultCode> y <faultName>, para identificar un error[14].

```
1  POST /RPC2 HTTP/1.1
2  Host: 132.248.181.156:50000
3  Accept-Encoding: gzip
4  User-Agent: xmlrpclib.py/1.0.1 (by www.pythonware.com)
5  Content-Type: text/xml
6  Content-Length: 439
7
8  <?xml version='1.0'?>
9  <methodCall>
10         <methodName>join_room</methodName>
11         <params>
12                 <param>
13                         <value><string>Minerocraft</string></value>
14                 </param>
15                 <param>
16                         <value><string>mine</string></value>
17                 </param>
18                 <param>
19                         <value><string>kiwilord</string></value>
20                 </param>
21                 <param>
22                         <value><string>kiwi</string></value>
23                 </param>
24                 <param>
25                         <value><base64>CgMS6A==</base64></value>
26                 </param>
27                 <param>
28                         <value><int>44309</int></value>
29                 </param>
30         </params>
31 </methodCall>
```

Figura 10: Ejemplo de una petición XML-RPC

2.11 Python

Python es un lenguaje de programación poderoso y fácil de aprender. Tiene estructuras de datos eficientes de alto nivel y una aproximación simple pero efectiva a la programación orientada a objetos. Su elegante sintaxis y tipificación dinámica, junto con su índole interpretado, lo vuelven un lenguaje ideal para scripting y un desarrollo rápido de aplicaciones en muchas áreas y plataformas.

El intérprete de Python puede ser extendido fácilmente con nuevas funciones y tipos de datos implementados en C, C++ u otros lenguajes que pueden ser llamados desde C. Python también es adecuado como un lenguaje de extensión para aplicaciones personalizables[15].

Ejemplo de “Hola Mundo” en Python desde una terminal:

```
>>> print “Hola Mundo”
Hola Mundo
```

2.12 Qt

Qt es el framework estándar para C++ para el desarrollo de software multiplataforma de alto nivel. Cuenta con licencias gratuitas GPL v3 y LGPL v2, además de una licencia comercial por parte de la empresa Digia.

Qt contiene una gran variedad de widgets o controles,

elementos visuales que se combinan para crear interfaces gráficas de usuario como botones, menús, mensajes de texto, etc.

También cuenta con administradores de diseño que ayudan a organizar widgets internos en el área de sus widgets que los contienen, posicionando y redimensionando automáticamente a los widgets internos. Los administradores de diseño ofrecen flexibilidad y velocidad de respuesta a las interfaces de usuario, habilitándolas para adaptarse cuando se actualicen estilos, orientaciones o tipografías.

Una característica que diferencia a Qt de otros frameworks es su mecanismo de programación reactiva llamado Signals and Slots que se utiliza para la comunicación entre objetos. En este mecanismo, una señal se emite cuando ocurre un evento. Un slot es una función que es llamada en respuesta a una señal en particular.

Un solo slot puede ser conectado a tantas señales como sea necesario y viceversa, incluso puede conectarse una señal a otra, lo cual emitirá la segunda señal inmediatamente después de la primera. Las conexiones entre las señales y los slots pueden ser agregadas o borradas en cualquier momento, y pueden ser realizadas entre objetos en diferentes hilos de ejecución[16].

PyQt es la biblioteca de Qt para Python v2.X y v3.X, tiene soporte para Qtv5 (PyQt5) y Qtv4 (PyQt4). Está disponible para todas las plataformas que soportan Qt bajo diferentes licencias incluyendo la GNU GPL v3 y también cuenta con una licencia comercial[17].

Ejemplo de “Hola Mundo” en PyQt:

```
import sys
from PyQt4 import QtGui
class ventana(QtGui.QMainWindow):
      def __init__(self):
            super(ventana,self).__init__()
            self.setWindowTitle("Ejemplo")
            self.label = QtGui.QLabel(self)
      def agregar_texto(self,texto):
            self.label.setText(texto)
app = QtGui.QApplication(sys.argv)
v = ventana()
v.show()
v.agregar_texto("Hola Mundo")
sys.exit(app.exec_())
```

Figura 11: Resultado de "Hola Mundo" en PyQt

3. Sistema Atún

"Saber qué es lo que sabes
y saber qué es lo que no sabes,
ese es el verdadero conocimiento."
Confucio

3.1 Descripción

Atún es un sistema de transporte de paquetes que permite simular una LAN, de forma distribuida, permitiendo que diversos equipos que se encuentren en una misma sala dentro del sistema puedan comunicarse entre si, aun cuando éstos se encuentren en diferentes ubicaciones geográficas; con esto poder utilizar entre ellos ciertas aplicaciones, o extensiones de ellas, que funcionen en una LAN.

3.2 Arquitectura

El sistema presenta una arquitectura de tipo Cliente – Servidor con topología de estrella. Es posible conectar varios clientes con un mismo servidor pero no lo es el conectar un cliente a varios servidores. Sin embargo, un solo equipo puede ejecutar a más de un cliente a la vez, permitiéndole conectarse a un servidor diferente con cada cliente en ejecución.

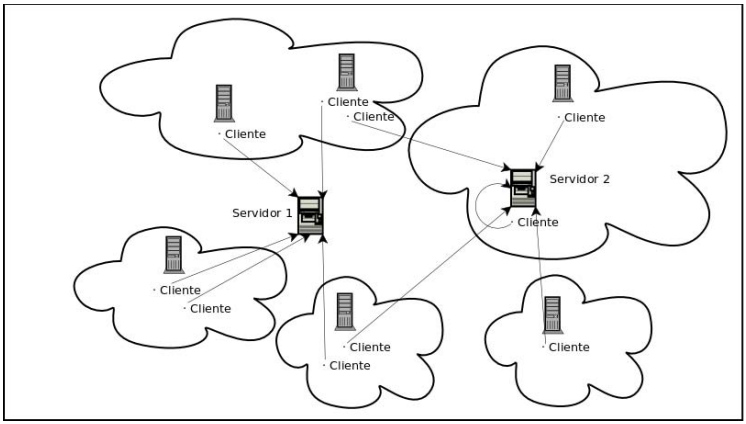

Figura 12: Arquitectura del sistema Atún

3.2.1 Servidor

El servidor puede ser en principio cualquier equipo, con la única restricción de que éste debe tener una dirección IP pública o, si se encuentra detrás de un NAT, éste le debe redireccionar los puertos necesarios.

Aquí el sistema aloja “salas” que representan esencialmente a cada una de las LAN virtuales que pueden ser administradas, las cuales cuentan con un identificador único que no depende de la dirección IP de las redes, por lo que es posible crear sin ningún problema varias salas con la misma dirección IP en un mismo servidor. Por esto mismo, el límite máximo de salas que pueden ser administradas por un servidor depende de las capacidades del equipo y la infraestructura que lo rodea y no de las del propio sistema.

La función principal de un servidor *Atún* es la de redireccionar los paquetes que se envían entre miembros de una misma sala, guardando sus extremos públicos y virtuales, pero también se encarga de informar a cada cliente de sus respectivos extremos para su propia configuración (la cual es necesaria si se encuentra detrás de un NAT) y también ayuda a los equipos a atravesar sus NAT para que éste no sea un impedimento en la transmisión de los datagramas.

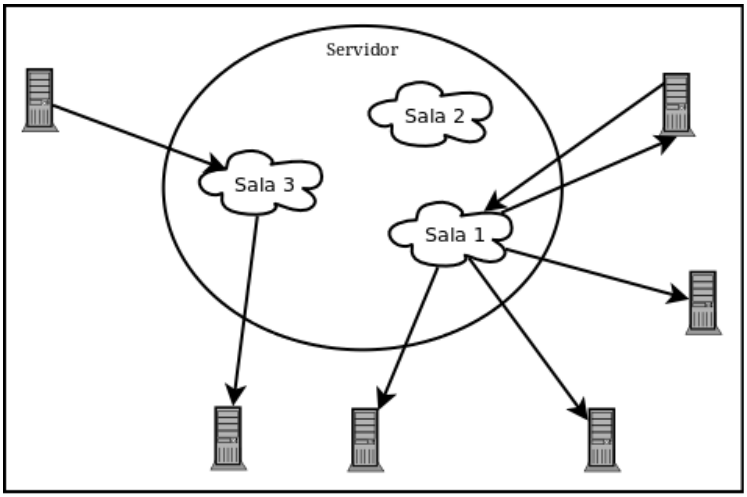

Figura 13: Funcionamiento del servidor de Atún

3.2.2 Cliente

Un cliente puede ser cualquier equipo que conozca el extremo público del servidor en el cual se reciben los paquetes de *Atún*.

Dado que en el equipo del cliente sí se usa la dirección IP para que el Kernel identifique a qué red pertenece cada una de sus interfaces de red y poder generar las rutas adecuadas, aquí si existe la restricción de que las distintas redes no pueden existir en los mismos rangos IP, y esta restricción es la que limita la cantidad de redes a las que puede conectarse un mismo equipo.

La función del cliente *Atún* es encargarse registrarse en el servidor y poder tener acceso a las salas existentes o crear nuevas salas. Luego se encarga de configurar las interfaces de red virtuales para cada sala a la que el usuario quiera conectarse. Además se encarga de crear a los “Xchangers” para que administren cada una de las redes desde el lado del cliente.

Un Xchanger es una entidad que se crea en el equipo del cliente, el cual se encarga de mover los paquetes que se envían y reciben entre el dispositivo virtual y el socket UDP del túnel de la sala. También se encarga de verificar que el canal de comunicación entre el cliente y el servidor se mantenga con vida desde que se entra hasta que se sale de una sala. Por último se encarga de cerrar el archivo del dispositivo para que automáticamente se elimine su interfaz virtual.

Los clientes cuentan con 2 canales de comunicación, uno brindado por XMLRPC para el manejo de los paquetes administrativos de las salas y el otro brindado por el túnel TUN para el flujo de los paquetes de datos de las aplicaciones.

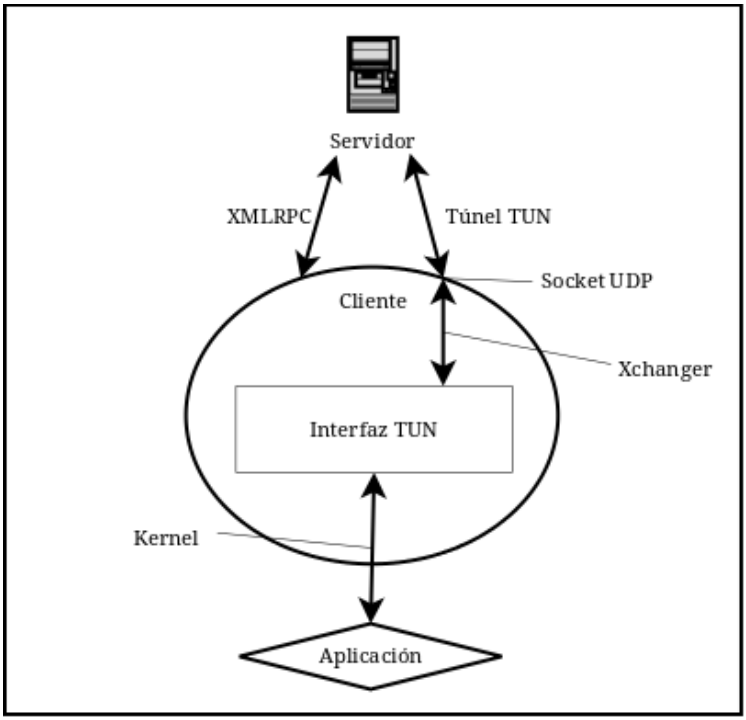

Figura 14: Funcionamiento del Cliente de Atún

3.3 Funcionamiento

El sistema en general se puede dividir en 3 fases importantes que se realizan para cada sala a la que se quiera conectar un cliente: configuración, mantenimiento y uso.

Antes que nada se necesita configurar el servidor para que pueda recibir y manejar las peticiones de los clientes. Primero se

abre un socket UDP el cual se encargará de enviar y recibir los paquetes de datos de las aplicaciones. Luego se crea un ejemplar de un servidor XMLRPC, al cual se le proporcionan una serie de procedimientos que sirven para la administración de las salas y después se quedará en espera de las peticiones de los clientes. Cada vez que el servidor reciba una petición, este verificará si se trata de una petición administrativa o de un paquete de datos y así delegar la responsabilidad al manejador correspondiente. Ambos canales se configuran en el mismo puerto pero el de XMLRPC lo hace por TCP y el túnel por UDP.

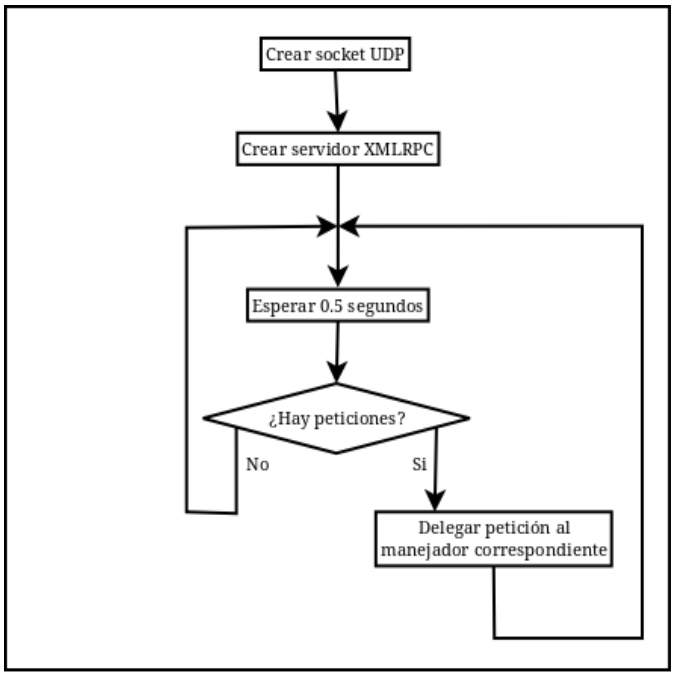

Figura 15: Configuración del servidor Atún

Para todas las operaciones que requieren datos proporcionados por el usuario, el cliente cuenta con una interfaz gráfica para que el usuario pueda capturarlos y enviarlos fácilmente.

Lo primero que debe hacer un usuario es proporcionarle al sistema los datos del extremo público del servidor, luego el cliente, apoyándose de XMLRPC, crea un proxy para poder comunicarse por él.

Luego de ésto comienza la fase de configuración. Es necesario registrar al usuario y agregarlo a la primera sala que desee. Para el registro se le pide al usuario que proporcione un nombre y una contraseña para poder identificarlo dentro de las salas. En la misma ventana se puede elegir entre crear una sala nueva o unirse a una existente.

Para crear una nueva sala, se debe proporcionar un nombre para la sala y una contraseña de acceso; la dirección IP de la LAN que se va a usar junto con su máscara de red para limitar la cantidad de usuarios y una clave de administrador para las acciones importantes de las salas como impedir accesos, expulsar usuarios o eliminar la sala por completo. El creador de una nueva sala se agrega automáticamente a la lista de administradores. (las funciones de los administradores no se implementaron para los objetivos de esta tesis). Los datos se envían al servidor por medio de XMLRPC y son revisados allá para evitar posibles errores o conflictos como salas duplicadas. Luego de crearse la sala en el servidor, se ejecuta la misma operación que se haría para unirse a una sala existente sin que el usuario tenga que volver a escribir los mismos datos.

Si se quiere unir a una sala ya existente, solo se necesita proporcionar el nombre y la contraseña de acceso. Antes de enviar los datos, el cliente envía una petición (*Echo)* al servidor con la cual

el servidor le reporta al cliente la información de su extremo público como si se quisiera comunicar consigo mismo usando Hole Punching. Ésta información es agregada a la proporcionada por el usuario y se envían todos los datos juntos para que el servidor pueda asociarlo a la sala deseada y responderle con la información necesaria para la configuración de su interfaz virtual, tales como: su extremo virtual asignado y el identificador único de la sala para poder rutear los paquetes de datos. Igualmente los datos son enviados al servidor por medio de XMLRPC y se revisan para evitar errores como usuarios duplicados o contraseña de acceso incorrecta.

No es necesario escribir en todos los campos de la ventana, solo en los datos de usuario y en el correspondiente para crear o unirse a una sala.

Para la configuración de la interfaz virtual, el cliente abre el archivo del dispositivo TUN y lo configura utilizando las banderas de TUN/TAP para indicar que se utilizarán datagramas UDP sin encabezados adicionales para crear un túnel. Ésto crea la interfaz virtual y después es configurada con los datos obtenidos del servidor y el nombre de la sala proporcionado anteriormente, con ayuda del comando *ifconfig* de Linux.

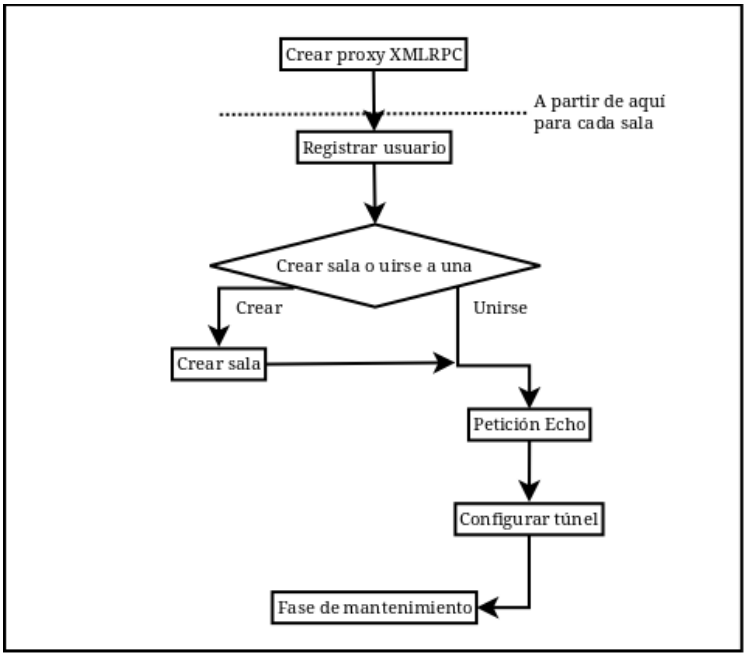

Figura 16: Configuración del cliente Atún

Si la configuración de la interfaz resulta sin problemas, entonces el túnel queda habilitado para empezar a comunicarse con el servidor y se crea un Xchanger para manejar el dispositivo y el socket UDP. Luego, el sistema pasa a la fase de mantenimiento que se encargará de mantener con vida al túnel, esta fase se explicará en el siguiente subtema.

La fase de uso comienza en cuanto una aplicación comienza a enviar paquetes de datos por la interfaz virtual. Para que la aplicación pueda comenzar a mandar paquetes a otros miembros de la sala, los paquetes deben ser enviados al extremo virtual de ellos;

con esto, el Kernel captura las solicitudes de envío de datagramas vía la interfaz virtual, y recoloca el datagrama en el archivo del dispositivo TUN, como si se tratase de un medio físico normal. El Xchanger que administra la red, se da cuenta de que hay algo que enviar en el dispositivo virtual. El Xchanger toma el datagrama del dispositivo y lo encapsula con las cabeceras correspondientes para indicarle al servidor que se trata de un paquete de datos que debe ser reenviado, y la sala en la que puede encontrar a su destinatario. El paquete resultante es enviado al servidor por medio del túnel.

El servidor, al recibir el paquete y observar que se trata de un reenvío, desencapsula el datagrama original y busca a su destinatario en su tabla de direcciones con ayuda del identificador de la sala proporcionado por el encabezado adicional, y el extremo virtual al cual está dirigido el datagrama. Si el cliente encuentra al destinatario solicitado, le envía el datagrama original a su extremo público utilizando el túnel que tiene con ese cliente.

El cliente del destinatario recibe el datagrama en el socket UDP del túnel y lo escribe en el archivo de su dispositivo TUN. El Kernel del sistema operativo lo toma como un datagrama recibido normalmente desde un dispositivo físico de red y se lo pasa a la aplicación que se encuentra esperando datagramas en la interfaz virtual.

En este punto, el transporte se da por finalizado y el sistema puede ya sea quedarse en la fase de uso, mientras los clientes siguen enviándose paquetes entre ellos, o puede volver a la fase de mantenimiento.

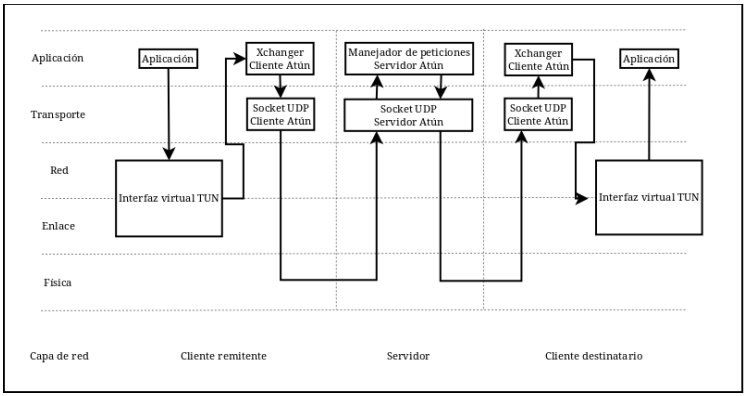

Figura 17: Flujo a través de las capas de red en la fase de uso

La interfaz gráfica separa cada sala en la ventana principal utilizando una pestaña con el nombre de la sala y el nombre de usuario dentro de ella. Además, cuenta con la opción de crear o unirse a otras salas y la opción de salir de ellas.

Para salir de una sala solo basta con oprimir el botón salir en la pestaña de la sala correspondiente. El cliente primero notifica al servidor por medio de XML-RPC para que borre al usuario de sus tablas y luego al Xchanger para que finalice su proceso. El Xchanger se encarga de cerrar tanto el archivo del dispositivo TUN para que la interfaz sea eliminada, como el socket UDP para cerrar el túnel.

Si se cierra completamente la ventana principal de la interfaz gráfica, automáticamente se les notifica a todos los Xchangers para que finalicen su proceso, cerrando todos los dispositivos y sockets sin notificar al servidor de la salida.

3.3.1 Funcionamiento de clientes detrás de un NAT

El principal propósito de la fase de mantenimiento es brindar al sistema con persistencia para el túnel entre el cliente y el servidor, ayudando a solucionar posibles problemas que pueden presentarse cuando el cliente se encuentra detrás de un NAT.

Cuando un cliente se encuentra detrás de un NAT, es posible que los NAT's puedan llegar a cerrar la sesión UDP del túnel por inactividad antes de que pueda comenzar la fase de uso del sistema, por lo que es necesario asegurar la supervivencia de la sesión para mantener la comunicación abierta con el servidor. En la fase de mantenimiento, mediante una de las técnicas para la supervivencia del agujero de *Hole Punching*, se intenta mantener actualizado el temporizador de inactividad del NAT para que éste no borre de sus tablas a la sesión UDP del túnel enviando peticiones (*Refresh*) vacías al servidor, para mantener activa la sesión, las cuales son identificadas y respondidas por el servidor, confirmando que la sesión sigue con vida.

Sin embargo, dado que el límite de tiempo de inactividad es diferente para cada NAT, algunos pueden llegar a borrar la sesión entre estas peticiones. Para solucionar el problema, se toman en cuenta una cierta cantidad de peticiones *refresh* consecutivas sin respuesta para pensar que la sesión ha sido borrada por inactividad y que se ha generado una nueva y se procede a enviar al servidor una petición (*Update*) pidiéndole que actualice su información sobre su extremo público en sus tablas, proporcionándole su extremo virtual y el identificador de la sala para ubicarlo. Si el servidor logra encontrar al usuario, procede a actualizar su información con los datos del extremo público obtenidos en el encabezado de la petición. Con ésto el cliente debe ser capaz de volver a recibir respuestas de sus peticiones de mantenimiento y paquetes de datos de los demás

miembros de la sala, reiniciando los contadores de peticiones sin respuesta.

De igual forma, se cuentan estas peticiones para renovar el extremo público del cliente que no se responden, y si se supera el límite se considera como un problema irrecuperable y se notifica al Xchanger para que finalice su proceso.

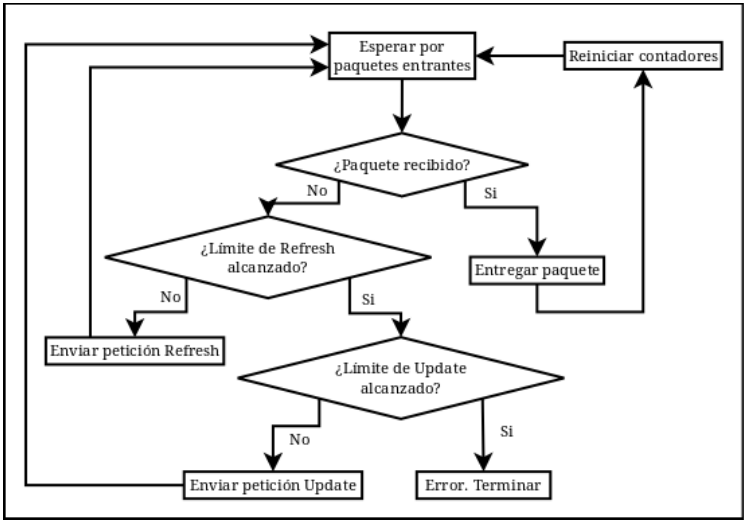

Figura 18: Flujo de paquetes en la fase de mantenimiento

3.4 Especificaciones Técnicas

Para la iniciación del servidor se enlaza el socket UDP a la dirección IP 0.0.0.0 para poder enlazarlo a todas las direcciones IPv4 disponibles para el servidor en caso de contar con más de una[18] y al puerto 50000 que no se encuentra reservado, pero si ya se encuentra en uso se puede elegir otro sin problema.

Los extremos públicos y virtuales de los clientes son almacenados tanto de forma numérica para realizar operaciones administrativas, como una cadena de caracteres binarios (de 4 bytes para las direcciones IP y de 2 bytes para los puertos) para darle prioridad a la velocidad (sobre el espacio) para retransmitir los paquetes.

Cada sala cuenta con una lista con los miembros que se encuentran en ella y el servidor con una lista de las salas que administra, así como una tabla de direcciones en la que se mapean los extremos virtuales con los públicos en cada sala de la siguiente manera:

[identificador de sala, [extremo virtual. [extremo público]]]

Dado que salas diferentes pueden existir en los mismos rangos IP el identificador de la sala es el que tiene mayor peso al momento de buscar el extremo público de alguien para la retransmisión de paquetes. Una vez identificada la sala se revisa el extremo virtual de los miembros para dar con el extremo público buscado.

Dado que los clientes necesitan privilegios de administrador para poder configurar las interfaces virtuales, requieren ser

ejecutados con éstos privilegios, de lo contrario el Kernel les impedirá el acceso a los comandos de configuración necesarios.

Para inicializar un cliente se utiliza la dirección 127.0.0.1 y el puerto 50000 como valores por defecto para fines de desarrollo, pero son cambiados por el usuario al principio de la configuración del cliente.

La interfaz gráfica cuenta con verificadores que utilizan expresiones regulares en todos los campos que requieren de una estructura en particular, como el formato de una dirección IP o un número de puerto que solo se puede elegir entre 1024 y 65535 que es el rango de los puertos no reservados, o simplemente que no sea vacío, como el nombre de la sala. La máscara de red acepta valores entre 8 y 24, pero se recomiendan solo 8, 16 o 24 que son los valores más comunes.

Una vez que todos los campos cumplan las condiciones que se les piden, se habilita el botón para enviar los datos al servidor.

Aunque el nombre de las salas no cuenta con un límite, se recomienda que sea de una longitud menor a 15 caracteres ya que el mismo nombre se usa para nombrar a la interfaz virtual y ese es el límite. Además, si el tamaño lo permite, al final del nombre se utiliza un formato para poder agregarle un valor numérico y así poder acceder a varias salas con el mismo nombre (solo posible si se ejecutan varios clientes en diferentes servidores ya que un mismo servidor no acepta repetir el nombre de las salas). Por ejemplo, si se accede a la sala “juegos” en 2 servidores distintos, en el equipo del cliente se crearán las interfaces virtuales juegos_1 y juegos_2.

Todas las llamadas realizadas por XMLRPC, de no haber algún error externo al sistema, devuelven información útil al usuario, un código de éxito y los resultados o sólo un código de error. En caso

de existir algún problema con los datos enviados, el código de error es interpretado por el cliente brindando la información necesaria para corregir el problema sin entrar en detalles. Si los datos enviados fueron correctos, el cliente simplemente procede a trabajar con los resultados.

Por su parte, a los paquetes de datos enviados a través del túnel se les agrega un encabezado que le ayudará al servidor a saber que hacer con ellos. Primero se le indica la longitud del datagrama original para que sepa cuanta información debe manejar. El segundo campo es un código de operación que indica el propósito del paquete. Por último, el tercer campo del encabezado es el identificador de la sala, éste campo es opcional ya que no todos los paquetes que recibe el servidor lo necesitan. Todos los campos son enteros de 4 bytes.

Byte inicial	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Longitud del paquete original																															
4	Opcode																															
8	Identificador de la sala																															

Figura 19: Encabezado adicional de Atún

Existen 4 códigos de operación diferentes:

Opcode 0. Echo. Se utiliza para informar al cliente sobre los datos de su extremo público. Esta operación no requiere de información adicional por lo que la longitud del paquete contenido es 0 y el identificador de la sala no es requerido. El servidor responde con la información de su extremo público.

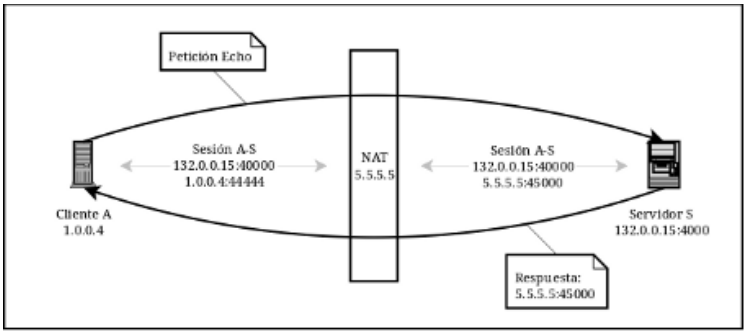

Figura 20: Petición Echo

Opcode 1. Forward. Se utiliza para retransmitir un paquete a otro miembro de la sala. La longitud del paquete la proporciona el datagrama original y el identificador de sala lo proporciona el Xchanger correspondiente. El servidor retransmite el paquete a su destinatario y no responde la petición al remitente.

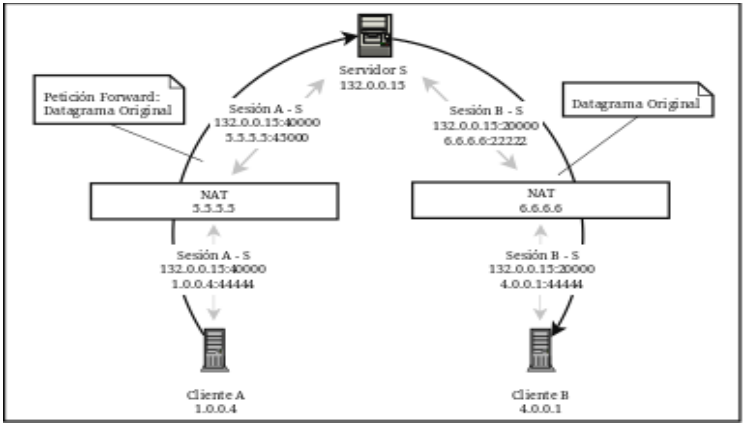

Figura 21: Petición Forward

Opcode 2. Refresh. Se utiliza para mantener viva la sesión UDP en el NAT del cliente. La longitud del paquete contenido es 0 y el identificador de la sala no es requerido. El servidor responde con otro paquete vacío. 5 paquetes no respondidos con este opcode (equivalentes a 15 segundos sin respuesta) suponen un cambio en la información de la sesión en el NAT.

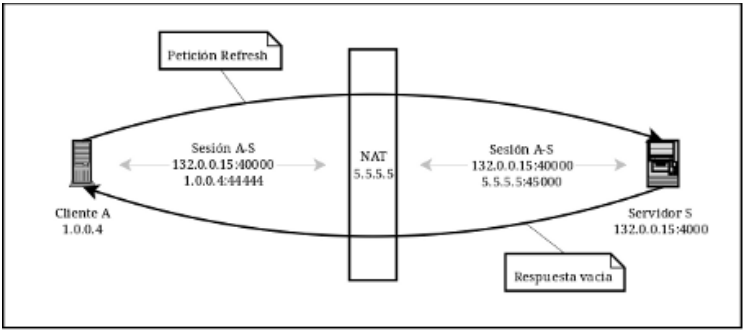

Figura 22: Petición Refresh

Opcode 3. Update. Se utiliza para actualizar la información de la sesión UDP en las tablas del servidor. La longitud del paquete contenido es 4 (bytes) correspondiente a la longitud del extremo virtual del cliente. El identificador de sala es requerido para encontrar la información. El servidor no responde a ésta petición pero con la información actualizada pude responder a las demás para reiniciar los contadores. 20 paquetes no respondidos con este opcode (equivalentes a 5 minutos sin respuesta) suponen un error irrecuperable.

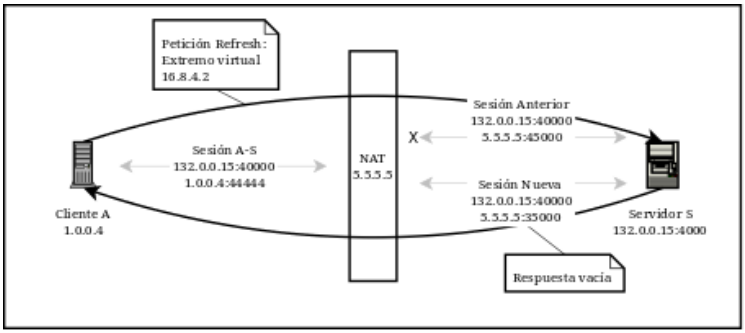

Figura 23: Petición Update

Si algún paquete llega al servidor con un opcode diferente a los anteriores, la operación se registra en el registro con la información del extremo origen y se descarta el paquete.

3.4 Ejemplo de uso

El objetivo de este ejemplo es mostrar que la comunicación entre equipos remotos es posible fácilmente con la ayuda del sistema *Atún*.

Se utilizó un programa llamado *netcat* para crear un chat entre 2 equipos remotos[19], uno en la Ciudad de México y el otro en Monterrey, utilizando Atún.

El primer paso es ejecutar el servidor en el cual se conectarán los clientes.

Figura 24: Ejecutando el servidor Atún

Una vez que el servidor está en ejecución se puede proceder a ejecutar los clientes, los cuales deben ejecutarse con privilegios de administrador o no se ejecutarán ya que no sería posible crear las interfaces virtuales.

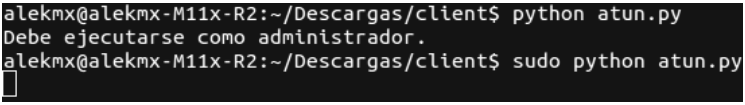

Figura 25: Ejecutando el segundo cliente Atún

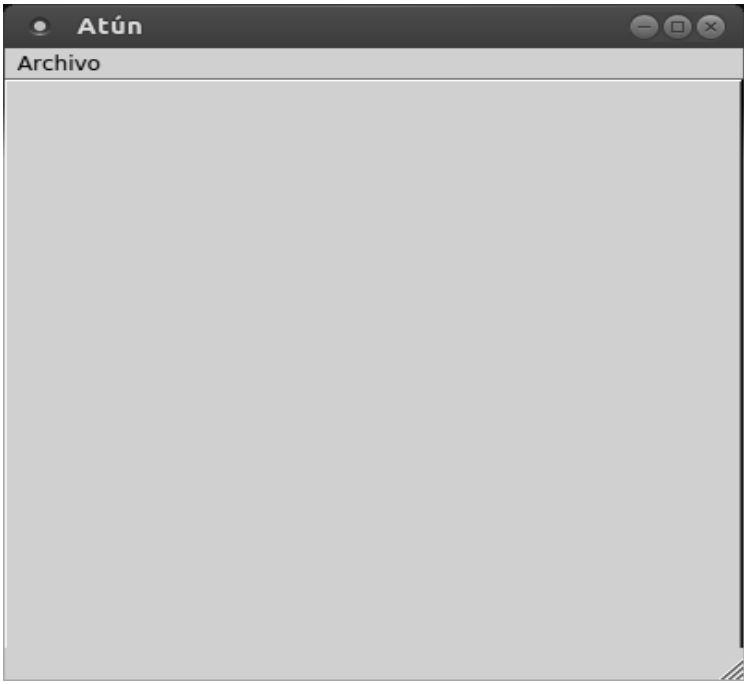

Figura 26: Ventana principal de Atún

Ahora hay que conectarse al servidor, para ello se abre el menú “Archivo” en la parte superior y luego en la opción “Nueva conexión”. Como es la primera vez que se abre una conexión, se necesita decirle al cliente la dirección del servidor.

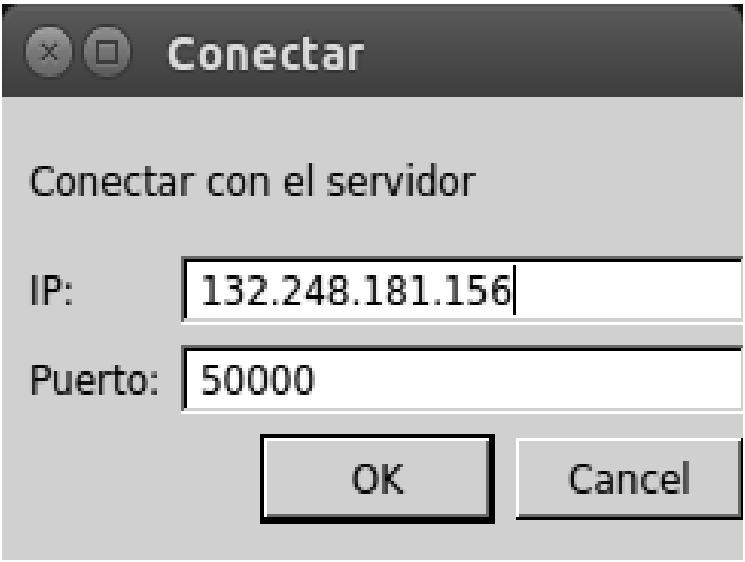

Figura 27: Dirección IP y puerto del servidor Atún.

Después es necesario que uno de los clientes cree una sala para que el otro se pueda unir a ella.

Figura 28: Para crear una sala se utilizan las secciones "Usuario" y "Crear".

Figura 29: Para unirse a una sala se utiliza la sección "Usuario" y "Unirse".

Si todo fue correcto ahora ambos clientes deberán tener una nueva interfaz de red con el mismo nombre que la sala.

```
kiwilord@Kiwilap:~$ ifconfig
Chat_0    Link encap:UNSPEC  direcciónHW 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          Direc. inet:4.4.0.1  P-t-P:4.4.0.1  Másc:255.255.0.0
          ACTIVO PUNTO A PUNTO FUNCIONANDO NOARP MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:500
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura 30: Interfaz virtual del primer cliente

```
alekmx@alekmx-M11x-R2:~/Descargas/client$ ifconfig
Chat_0    Link encap:UNSPEC  direcciónHW 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          Direc. inet:4.4.0.2  P-t-P:4.4.0.2  Másc:255.255.0.0
          ACTIVO PUNTO A PUNTO FUNCIONANDO NOARP MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:500
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura 31: Interfaz virtual del segundo cliente

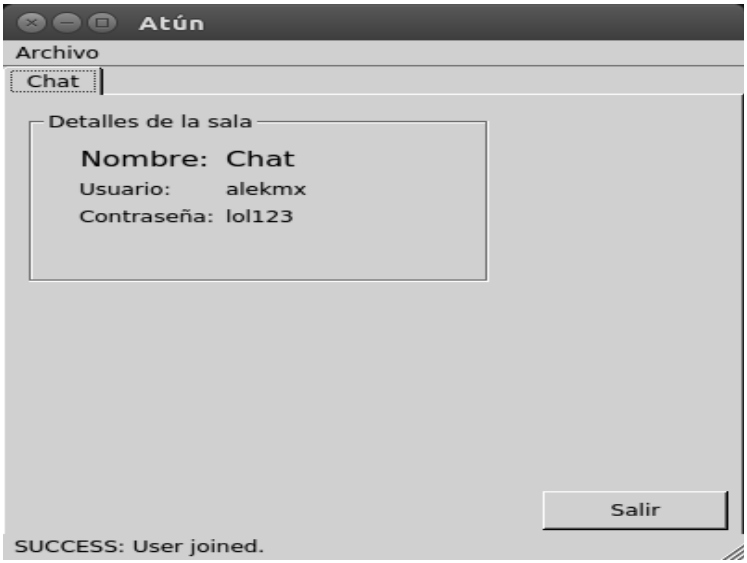

Figura 32: Ventana principal de Atún del segundo cliente despues de entrar a la sala

Una vez que se tienen las interfaces virtuales el sistema pasa a la fase de mantenimiento donde el cliente comienza a enviar las peticiones *Refresh* para mantener activa la sesión UDP. En la

siguiente imagen se puede ver el tráfico del servidor, algunas peticiones Refresh del primer cliente y también se ve una petición *Echo* del segundo cliente que estaba por entrar.

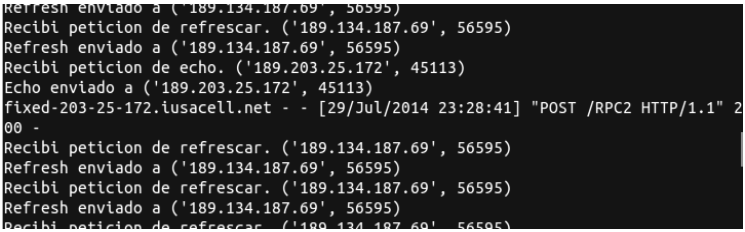

Figura 33: Fase de mantenimiento del primer cliente vista desde el servidor

Se utilizaron 2 terminales ejecutando *netcat* para simular una sala de chat entre los dos clientes. En la primera terminal se ejecuta de la siguiente manera:

nc -ul <extremo virtual local> 50000

Donde <extremo virtual local> es la dirección IP que se le asignó a la interfaz virtual propia del cliente. Ésto permite escuchar mensajes entrantes en el puerto UDP 50000 de la interfaz virtual. En la segunda terminal se ejecuta *netcat* así:

nc -u <extremo virtual remoto> 50000

Donde <extremo virtual remoto> es la dirección IP que se le asignó a la interfaz virtual del cliente con el que se va a comunicar. Esto permite escribirle mensajes a la dirección virtual del cliente remoto en el puerto 50000 de UDP.

En el caso del primer cliente cuya dirección virtual asignada fue 4.4.0.1, los comandos quedarían de la siguiente manera.

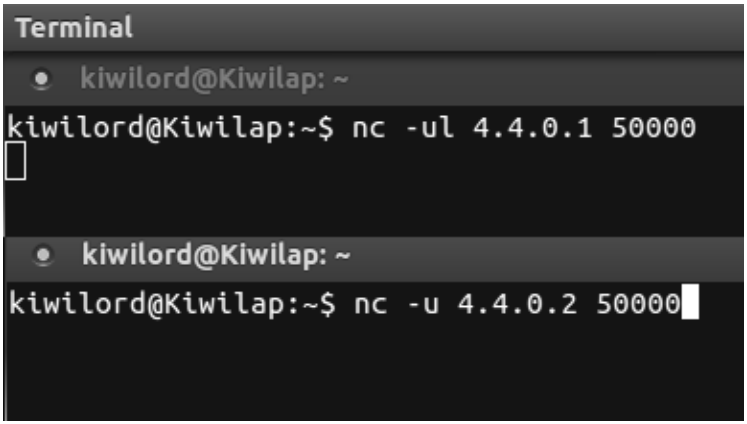

Figura 34: Comandos de netcat para simular una sala de chat

Con esos comandos, todo lo que se escriba en la terminal que hace referencia al cliente remoto, si el canal de comunicación funciona, lo debe poder ver su usuario en la terminal que se dedica a escuchar en su dirección local como se muestra en la siguiente imagen.

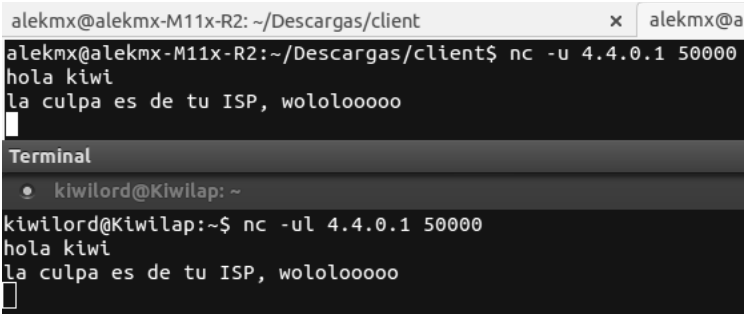

Figura 35: Chat por netcat funcionando gracias a Atún. (arriba) Segundo cliente como emisor y (abajo) primer cliente como receptor.

La imagen anterior muestra que se pudo tener una conversación entre los clientes[20] utilizando *netcat* gracias a la ayuda del sistema *Atún*, ya que sin las interfaces creadas y administradas *netcat* no permite escuchar en la dirección 4.4.0.1 puesto que no se tiene esa dirección en ninguna interfaz.

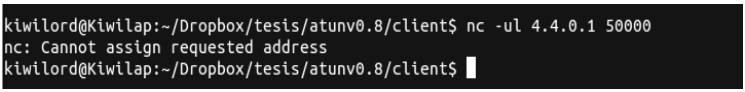

Figura 36: Netcat sin poder asignar la dirección despues de cerrar el cliente.

3.5 Prueba de eficiencia

El ejemplo anterior muestra que es posible la comunicación entre 2 clientes remotos pero ¿Qué tan eficiente es?

Para hacer una medición del desempeño del sistema se utilizó el programa *ping* con los mismos equipos remotos para enviar paquetes en 4 escenarios diferentes y observar los resultados. El comando con el que se ejecutó el programa en cada escenario es el siguiente:

ping -c 100 -s 84 <destino>

Donde -c 100 indica que son 100 paquetes los que se enviaron, -s 84 indica que el tamaño de los paquetes fue de 84 bytes que es el tamaño promedio de un paquete generado por un juego[21] y <destino> indica la dirección a la que se enviaron los paquetes.

El primer escenario utilizado fue el de enviar paquetes a la dirección local 127.0.0.1 para obtener una idea del mejor desempeño que podría tener *ping*. Comando:

ping -c 100 -s 84 127.0.0.1

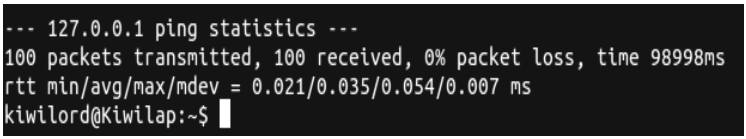

Figura 37: Resultado de hacer ping a la dirección local 127.0.0.1

La imagen muestra un 100% de paquetes recibidos en un total de 98998 ms.

El segundo escenario utilizado fue el de enviar paquetes a la dirección virtual 4.4.0.2 sin que se encuentre en ejecución el cliente *Atún* y poder medir el peor desempeño que podría tener *ping* en un escenario en el que los paquetes no puedan alcanzar su destino. Comando:

ping -c 100 -s 84 4.4.0.2

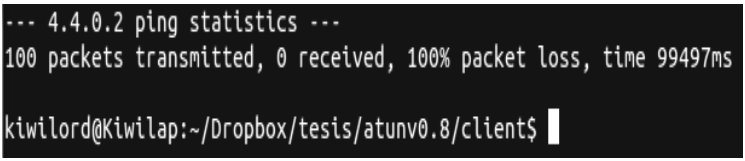

Figura 38: Resultado de hacer ping a la dirección virtual remota 4.4.0.2 que aun no existe por no encontrarse Atún en ejecución

La imagen muestra un esperado 0% de paquetes recibidos en un total de 99497 ms.

Suponiendo que el tiempo del primer escenario, de 98998 ms, indica un 100% de eficiencia y el tiempo del segundo escenario, de 99497 ms, indica un 0% de eficiencia, el tercer escenario se utilizó para medir la eficiencia que tiene el sistema *Atún* enviando paquetes a la dirección 4.4.0.2 teniendo ya la conexión brindada por el sistema. Comando:

ping -c 100 -s 84 4.4.0.2

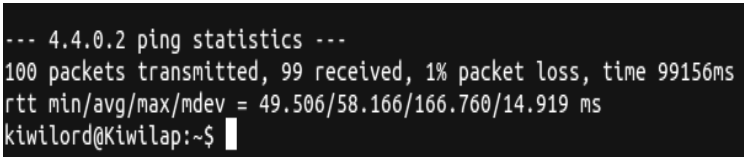

Figura 39: Resultado de hacer ping a la dirección 4.4.0.2 dentro de la misma red de área local virtual que existe gracias al sistema Atún

La imagen muestra un 99% de paquetes recibidos en un total de 99156 ms.

Para calcular la eficiencia del sistema se utilizó la siguiente fórmula:

$$Eficiencia = \frac{PeorTiempo - Valor}{PeorTiempo - MejorTiempo} * \%Recibidos$$

Donde MejorTiempo y PeorTiempo son los tiempos de los resultados del primer y segundo escenario respectivamente, Recibidos es el porcentaje de paquetes recibidos y Valor es el tiempo del resultado al que se le quiere medir su eficiencia.

Sustituyendo valores obtenemos:

$$Eficiencia = \frac{99497\,ms - 99154\,ms}{99497\,ms - 98998\,ms} * 99\,\%$$

$$= \frac{343\,ms}{499\,ms} * 99\,\% = 0.6874 * 99\,\% = 68.05\,\%$$

Por lo que la eficencia de la red de área local virtual de *Atún* es de 68.05% con respecto al mejor escenario.

El cuarto escenario se utilizó para hacer otra comparativa con el mejor escenario, pero esta vez enviando paquetes a otro miembro de la red local normal, en este caso la IP del equipo en la red local es: 192.168.1.80 y el segundo miembro es 192.168.1.74. Comando:

ping -c 100 -s 84 192.168.1.74

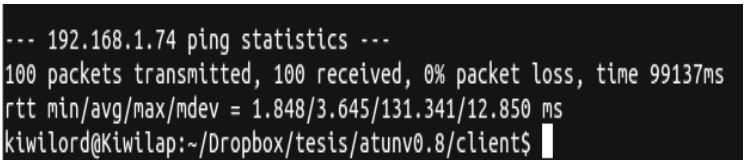

Figura 40: Resultado de hacer ping a la dirección 192.168.1.74 dentro de la misma red de área local normal

Utilizando la misma fórmula para calcular la eficiencia obtenemos:

$$Eficiencia = \frac{99497\,ms - 99137\,ms}{99497\,ms - 98998\,ms} * 100\,\%$$

$$= \frac{360\,ms}{499\,ms} * 100\,\% = 0.7214 * 100 = 72.14\,\%$$

Por lo que la eficiencia de una red de área local normal es de 72.14% con respecto al mejor escenario.

Los resultados obtenidos con estas últimas pruebas muestras que la red de área local virtual proporcionada por el sistema *Atún* es tan solo 4.09% menos eficiente que una red de área local normal.

4. Conclusiones

"Cada nuevo comienzo
viene del final
de algún otro comienzo"
Séneca

El sistema *Atún* es capaz de interconectar equipos remotos en una red LAN sin que los usuarios sufran dificultades sustanciales durante la configuración, lo cual abre las puertas a programas que hagan uso de redes locales para que éstos puedan ser utilizados de manera distribuida, usando software libre.

Para la creación del sistema fue necesario comprender un concepto fundamental de redes de computadoras, que es el del encapsulamiento, para el envío de datos IPv4 dentro de mensajes UDP, así como la forma en la que funcionan los NAT's para la implementación de la Técnica de Hole Punching, para permitir enviar dichos mensajes entre equipos que se encuentran detrás de ellos, con la ayuda de un tercer equipo. Y en general para todos los mensajes fue necesario comprender cómo opera el protocolo IPv4 y cómo se compone el formato de los encabezados de sus datagramas.

El hecho de separar las responsabilidades de las diversas capas de red resultó ser bastante útil ya que al solamente reenviar mensajes de la capa de red se pueden redirigir datos de las capas superiores, independientemente de los protocolos que éstas utilizan.

Python al parecer fue una buena elección para la implementación del sistema *Atún* ya que cuenta con API's para sockets muy fáciles de utilizar, así como para la manipulación de datos de bits.

Aunque el servidor de *Atún* requiere de una IP pública para funcionar correctamente, puede ser utilizado en un equipo personal que no posea una, siempre y cuando sea posible configurar su ruteador que lo provee de acceso a Internet, para que le redirija los paquetes que lleguen al puerto correspondiente al sistema.

El sistema *Atún* cuenta con las herramientas básicas para la creación de una red LAN; sin embargo, aún no se cuenta con las herramientas administrativas para la gestión de las salas una vez creadas, ni con elementos de seguridad como la codificación de los datagramas. Otro detalle del sistema es que usa rutas y comandos propios de un entorno Linux para la creación de las interfaces virtuales, por lo que por el momento es el único sistema operativo en el que puede funcionar.

Crear el sistema *Atún* fue personalmente muy gratificante ya que pude aprender cosas nuevas y repasar temas que no tenía del todo claros. Conforme iba avanzando en la implementación fueron surgiendo nuevos retos que me hicieron ver que un sistema así no era tan sencillo como se creía en un principio.

Aún cuando se logró cumplir el objetivo del sistema, se espera poder darle continuación más adelante, para así poder proporcionarle los elementos administrativos y de seguridad que le

faltan para ser un sistema completo, además de darle una mayor portabilidad.

Apéndice A. Glosario

DHCP. Dynamic Host Configuration Protocol. Protocolo de red el cual permite a un cliente de una red obtener sus parámetros de configuración.

Espacio de usuario. Se refiere al código que no forma parte del Kernel del sistema operativo o al almacenamiento crítico del sistema de archivos.

Extremo virtual. Término particular del sistema *Atún* que hace referencia a la dirección IP virtual de un cliente. A diferencia de los extremos públicos y privados, un extremo virtual no cuenta con un puerto de la capa de transporte ya que no es necesario.

Handshaking. Proceso automático de negociación mediante el cual se establecen de forma dinámica los parámetros de un canal de comunicación entre dos entidades antes de que comience la comunicación normal por el canal. TCP utiliza un Handshaking de 3 pasos para establecer una conexión.

ISP. Internet Service Provider. Es una organización que provee servicios de Internet.

Kernel. Parte fundamental del sistema operativo. Es el principal responsable de gestionar los recursos, a través de servicios de llamada al sistema.

Intranet. Red informática interna de una empresa u organismo, basada en los estándares de Internet.

GNU GPL. GNU General Public License. Es la licencia de software libre más ampliamente usada, que le garantiza al usuario final la libertad de usarlo, estudiarlo, compartirlo y modificarlo.

LAN. Local Area Network. Una red de área local es una red de computadoras concentrada en una misma área geográfica, como un edificio o una facultad.

Proxy. Un programa o dispositivo que realiza una acción en representación de otro.

Scripting. Un lenguaje de scripting es un lenguaje de programación que acepta scripts o guiones, programas escritos para un entorno de ejecución en particular que pueda interpretar (en lugar de compilar) y automatizar la ejecución de tareas que de otra forma podrían ser ejecutadas de una en una por un operador humano.

Bibliografía

[1]
Valve. 2012. "*Counter Strike: Global Offensive*". http://store.steampowered.com/app/730/ [Última consulta: enero, 2015].

Microsoft® Studios. 2000. "*Age of Empires: The Conquerors*". http://www.ageofempires.com/ [Última consulta: enero, 2015].

Ubisoft® Nadeo. 2011. "*Trackmania Canyon*". http://store.steampowered.com/app/232910/ [Última consulta: enero, 2015].

[2]
Jonas Wagner. 2007. "*Lanshark*". http://lanshark.29a.ch/en/About.html [Última consulta: enero, 2015].

Qualia. 2012. "*LAN Messenger*". http://lanmsngr.sourceforge.net/index.php [Última consulta: enero, 2015].

[3]
LogMeIn Hamachi®. Copyright © 2013. "*Guía de inicio*". https://secure.logmein.com/welcome/documentation/ES/pdf/Hamachi/LogMeIn_Hamachi_GettingStarted.pdf [Última consulta: junio, 2014].

[4]
Postel, J. (ed.). 1981. "*Internet Protocol – DARPA Internet Program Protocol Specification*". RFC 791. http://www.ietf.org/rfc/rfc791.txt

[Última consulta: Julio, 2014].

[5]
James F. Kurose, Keith W. Ross. 2012. "*Computer Networking: A Top Down Aproach*". 5A Edición. Pearson. 0-273-76896-4.

[6]
Internet Society. "*World IPv6 Launch*". http://www.worldipv6launch.org [Última consulta: junio, 2014].

[7]
Google. "*Google – IPv6. Estadísticas*". https://www.google.com/intl/es/ipv6/statistics.html [Última consulta: junio, 2014].

[8]
Microsoft TechNet. 2001. "*Virtual Private Networking: An Overview*". http://technet.microsoft.com/en-US/library/bb742566.aspx [Última consulta: junio, 2014].

[9]
Bryan Ford, Pyda Srisuresh, Dan Kegel. 2005. "*Peer-to-Peer Comunication Across Network Address Translators*". http://www.brynosaurus.com/pub/net/p2pnat/ [Última consulta: junio, 2014].

[10]
P. Srisuresh, M. Holdrege. 1999. "*IP Network Address Translator (NAT) Terminology and Considerations*". RFC 2663. http://tools.ietf.org/html/rfc2663 [Última consulta: Julio, 2014].

[11]
C. Jennings, F. Audet (Ed.). 2007. "*Network Address Translation (NAT) Behavioral Requirements for UNICAST UDP*". RFC 4787.

[12]
Maxim Krasnyansky. 1999 "*Universal TUN/TAP device driver*". https://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt [Última consulta: Julio, 2014].

[13]
Andrew D. Birrell, Bruce Jay Nelson. 1983. "*Implementing Remote Procedure Calls*". http://www.cs.princeton.edu/courses/archive/fall03/cs518/papers/rpc.pdf [Última consulta: Julio, 2014].

[14]
Dave Winer. 1999. "*XML-RPC Specification*". http://xmlrpc.scripting.com/spec.html [Última consulta: Julio, 2014].

[15]
Python Software Fundation. 1999-2014. "*The Python Tutorial*". https://docs.python.org/2.7/tutorial/index.html [Última consulta: Julio, 2014].

[16]
Qt Project. 2013. "*Qt Whitepaper*". http://qt-project.org/wiki/QtWhitepaper [Última consulta: Julio, 2014].

[17]
Riverbank. 2013. "*What is PyQt?*". http://www.riverbankcomputing.co.uk/software/pyqt/intro [Última consulta: Julio, 2014].

[18]
Stackoverflow question. 2011. "*Python Socket bind to any IP?*". http://stackoverflow.com/questions/8033552/python-socket-bind-to-any-ip [Última consulta: Enero, 2015].

[19]
Linoxide. 2013. *"How To Create A Simple Chat With netcat In Linux"*. http://linoxide.com/tools/simple-chat-netcat-linux/ [Última consulta: Enero, 2015].

[20]
"*La culpa es de tu ISP, wololooooo*" referencia a mensajes de Age of Empires II.

[21]
Kuan-Ta Chen, Polly Huang, Chin-Laung Lei. "*Game Traffic Analysis: An MMORPG Perspective*". http://www.iis.sinica.edu.tw/~swc/pub/game_traffic_analysis.html [Última consulta: Enero 2015].