

# Documentación Servidor DNS

Fernández Santiago, Olarte Fabián, Vásquez Andrés

Pontificia Universidad Javeriana

Facultad de Ingeniería

Bogotá, Colombia

sa.fernandez@javeriana.edu.co

olarte\_fabian@javeriana.edu.co

af.vasquezr@javeriana.edu.co

**Abstract—** Este documento corresponde a la documentación del primer proyecto de la clase de Comunicaciones y Redes relacionado con el diseño e implementación de un servidor DNS, acogido al estándar RFC-1035, emitido y revisado por la organización IETF construido bajo la arquitectura cliente-servidor.

## I. INTRODUCCIÓN

Un servidor DNS es un componente esencial en la estructura de las comunicaciones de la actualidad, ya que provee a un sistema computacional y a sus diferentes aplicaciones la dirección ip exacta por la cual buscar o solicitar contenido en la red. Esto es posible mediante el protocolo DNS de la capa de aplicación, cuyo principal objetivo es proveer respuestas a consultas a partir de un dominio dado, con el fin de que el servidor DNS asignado devuelva o responda con la ip de dicho dominio bajo un cierto formato determinado que será explicado en los siguientes apartados. Cabe resaltar que el diseño e implementación de este protocolo cumple los requisitos del estándar descrito en el RFC-1035 provisto por la organización IETF.

## II. DESCRIPCIÓN TEÓRICA DEL PROTOCOLO

Es un protocolo que permite localizar direcciones IP con base a las consultas que pueda realizar un usuario. Su funcionamiento consiste básicamente en resolver una consulta que contenga un nombre o un alias de alguna página web, máquina o servicio relacionado, enviándole de vuelta al cliente la dirección IP que esté relacionada con aquella consulta realizada anteriormente, por ejemplo, si un cliente envía una consulta que contenga el nombre `www.google.com`, el servidor DNS se encarga de devolverle la dirección IP `8.8.8.8`, la cual le permite acceder a la página web de Google, lo mismo pasa con todas las páginas web o servicios relacionados, es decir, que sin el protocolo DNS, internet no podría funcionar [1] [2].

Estos servidores DNS tienen un tipo de bases de datos (master file) que contiene las direcciones IP, lo cual le permite resolver las consultas realizadas por los clientes. Cuando llega el caso de que el DNS en su master file no posea una dirección IP solicitada por una consulta del cliente, este puede realizar una consulta iterativa o recursiva, las cuales se basan generalmente en contactar a otros servidores DNS referenciados (foreign resolver) para que puedan resolver la consulta realizada, y posteriormente el DNS que actuó como cliente (preguntando a otros DNS) guarda en su caché la información para que pueda ser consultada posteriormente sin necesidad de contactar con otros DNS.

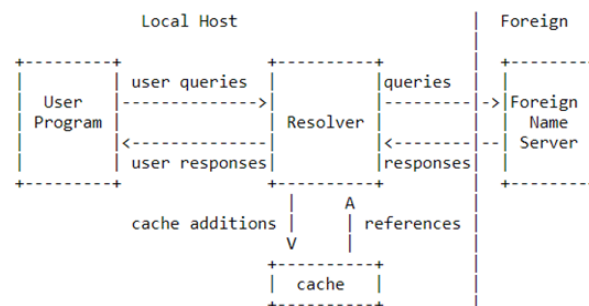


Fig. 1 Arquitectura de un servidor DNS

### A. Elementos de funcionamiento de la capa de aplicación

La capa de aplicación es la última capa de los modelos OSI y TCP/IP, es la capa la cual está más cerca en la interacción con el usuario (Cliente / Servidor), proporcionando servicios de red a las aplicaciones del cliente. Como se mencionó anteriormente, esta capa al ser la última de los modelos mencionados, es soportada por las capas anteriores a esta (figura 2), las cuales no prestan un servicio directo a las aplicaciones del cliente como si lo hace esta capa de aplicación [3].

El funcionamiento de esta capa se basa generalmente en hacer uso de protocolos para poder suministrar servicios como correo electrónico, world wide web, multimedia, procesamiento de texto, etc. Algunos protocolos usados son: DNS el cual resuelve consultas de los clientes (explicada con mayor detalle anteriormente), NIS el cual proporciona un control centralizado para poder administrar la red, NFS el cual proporciona servicios de archivos, RIP el cual es un protocolo de enrutamiento, TELNET el cual permite la comunicación entre terminales, HTTP el cual se utiliza para transferir archivos que conforman páginas web, FTP el cual se utiliza para la transferencia de archivos entre terminales, POP el cual es utilizado para recuperar correo electrónico de un servidor remoto, entre muchos otros [3] [4].



Fig. 2 Modelo OSI y TCP/IP

### B. Aplicación de protocolo UDP de capa de transporte en protocolo DNS

La capa de transporte tiene como tarea suministrar el servicio de transporte entre las capas, con el objetivo de que haya integridad en la información que es transportada entre dos máquinas. Esta capa es de gran importancia ya que sin esta los protocolos no podrían funcionar.

El protocolo UDP fue el que usamos en este proyecto, es el que se encarga de proporcionar una forma de enviar datagramas IP para la comunicación por intercambio de paquetes entre terminales, además cabe resaltar que este protocolo no es orientado a la conexión (NOC), es decir, que no hay confirmación de conexión, y por ende no se garantiza la entrega, mensajes duplicados ni tampoco el orden de transferencia, aunque es importante resaltar que en este proyecto se hizo uso del puerto 53 (utilizado en el protocolo DNS) el cual garantiza que la información sea enviada en el orden correcto). Este protocolo ayuda a la capa de aplicación en el proceso de transferencia de mensajes, y se hace a través de un formato (figura 3) [3].



Fig. 3 Formato UDP

El puerto de origen hace referencia a la dirección origen del cual sale la información, por lo cual se asume que la respuesta sea enviada a ese mismo puerto, en dado caso de que no se fije un puerto de origen este toma un valor de 0. El puerto de destino hace referencia a la dirección destino del cual llega la información. El campo de longitud contiene la suma de todos los bits del datagrama, lo cual implica que el mínimo valor que este campo pueda tomar es de 0. El campo de CheckSum contiene la suma de las palabras que contiene la constitución de una pseudo cabecera, y sirve para verificar que la información no ha cambiado en el proceso de transporte, ya que se dice que si este valor cambio fue por una alteración de la información. Y por último tenemos la pseudo cabecera la cual contiene la información de la dirección de origen, destino, protocolo y la longitud UDP (figura 4) [5].

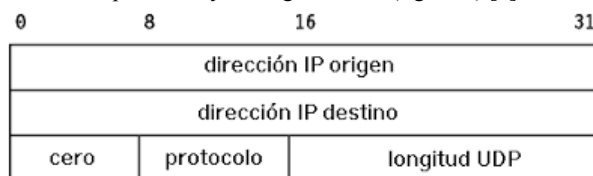


Fig. 4 Pseudo-cabecera UDP

### C. Constitución del header, question y answer

Las comunicaciones de las que hemos hablado anteriormente en la capa de transporte y aplicación se hacen a través de mensajes, los cuales tienen un formato definido (figura 5). Además, cabe mencionar que cada campo del formato del mensaje tiene otro formato.

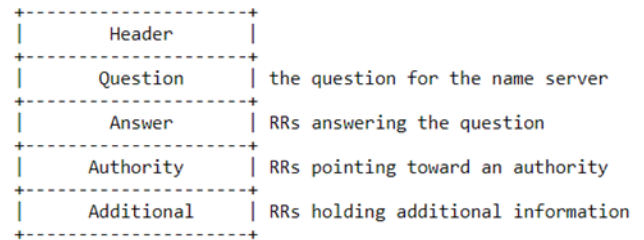


Fig. 5 Formato del mensaje DNS

El header se encarga de especificar si el mensaje en curso es una Answer o una Query (explicados posteriormente), y tiene el siguiente formato:

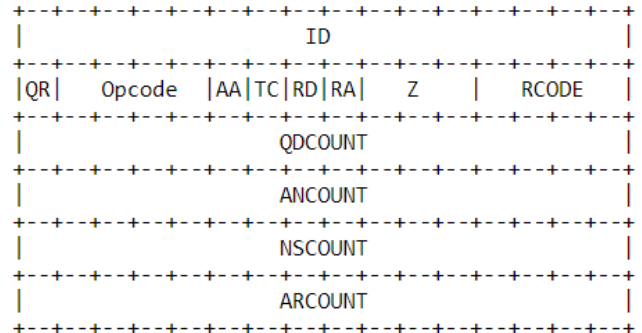


Fig. 6 Formato del header

En los campos del header tenemos: El ID el cual es generado automáticamente, se copia también en la respuesta con el fin de que el solicitante pueda identificar la respuesta de su correspondiente consulta. En el campo de QR puede haber 2 posibles números, 0 si el mensaje es de tipo consulta, o 1 si el mensaje es de tipo respuesta. En el siguiente campo, OPCODE, tiene 4 bits para poder definir el tipo de la consulta, pudiendo tomar diferentes valores, en el caso de 0 define si es una consulta estándar, en el caso de 1 indica si es una consulta inversa, el valor de 2 indica una solicitud de estado del servidor y los valores de 3 a 15 están reservados para su uso posterior. Además, este valor también se copia en la respuesta, el valor en la primera posición. El campo AA, indica si es una respuesta autorizada, es decir si la respuesta fue suministrada por el DNS que tiene autoridad para el nombre del dominio, en caso de que no sea una respuesta autorizada, se entiende que la respuesta fue suministrada por otro DNS que tiene autoridad para responder. El campo TC determina si el mensaje se truncó por temas de sobrepaso de longitud permitida en el canal de transmisión. El campo RD indica una solicitud al DNS para que realice la consulta de forma recursiva, además este valor también se copia en la respuesta. El siguiente campo, RA, informa si el DNS tiene consultas recursivas disponibles. El campo Z está reservado para un uso posterior, este debe tomar valor cero en las consultas y respuesta. El campo de RCODE hace referencia a las posibles respuestas que puedan surgir, estas respuestas varían dependiendo el valor, si toma el valor de 0 quiere decir que no hubo condición de error, el valor de 1 significa que hubo un error de formato y por ello el DNS no pudo interpretar la consulta, el valor de 2 indica que el DNS no pudo realizar la consulta debido a un problema, el valor de 3 indica que la referencia a la consulta realizada no existe, el valor 4 indica que el DNS no admite el tipo de consulta realizada por no estar implementada, el valor de 5 indica que el DNS rechazó la consulta y los valores correspondientes del 6 al 15 están reservados para un uso futuro. El siguiente campo, QDCOUNT, de 16 bits, indica el número de entradas en la sección de preguntas. El campo

ANCOUNT, también de 16 bits, indica el número de registros de recursos de la sección de respuesta. El campo NSCOUNT de 16 bits, indica el número de registros de recursos del servidor en la sección de registros de autoridad. El último campo ARCOUNT de 16 bits, indica el número de registros de recursos adicionales [6].

Una vez explicado el formato del header, explicaremos a continuación el formato de la sección de Question, la cual indica los parámetros que se van a preguntar.

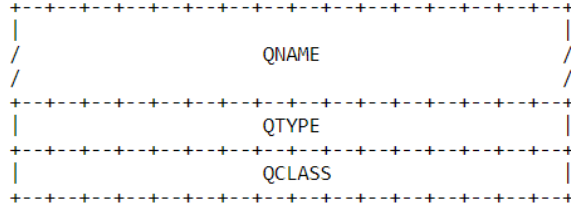


Fig. 7 Formato de la sección question

En el formato Question encontramos los siguientes campos: El campo de QNAME de 16 bits, define el nombre del dominio como una secuencia de etiquetas, la cual consta de un octeto de longitud seguido del número de octetos, el nombre del dominio termina con un octeto de longitud 0 para identifica la etiqueta nula de la raíz. El campo QTYPE de 16 bits, posee un código de dos octetos el cual especifica el tipo de consulta. Por último, tenemos el campo QCLASS, también de 16 bits, el cual está conformado por un código de dos octetos que especifica la clase de consulta [6].

Por último, tenemos los 3 campos de Answer, Authority y Additional, los cuales comparten el mismo formato (figura 8) y será explicado cómo fue hecho con los dos campos anteriores.

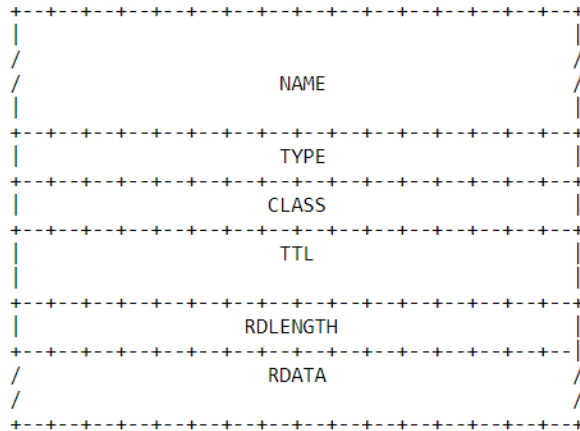


Fig. 8 Formato de la sección answer

En este formato tenemos los siguientes campos: El campo NAME define el nombre de dominio al que pertenece este registro de recursos. En el campo TYPE de 16 bits, se especifica el significado de los campos RDATA. En el campo CLASS se define la clase de datos en el campo RDATA. En el campo TTL de 32 bits, se especifica el tiempo en segundos, que el registro puede almacenarse en caché antes de descartarse, en caso de que sea 0 implica que el RR solo se puede utilizar para la transacción y no se debe almacenar. En el campo RDLENGTH de 16 bits, se establece el tamaño de octetos del campo de RDATA. Y por último tenemos el campo RDATA, el cual es de longitud variable dependiendo el tipo y la clase de registro de recursos [6].

### III. DISEÑO DEL APLICATIVO

El aplicativo fue construido según el funcionamiento de la arquitectura cliente-servidor exigida en los lineamientos del

trabajo, este está programado bajo el lenguaje de programación Java con elementos adicionales de base de datos, diseñada e implementada en el entorno MySQL con los elementos asociados al Master File, que tienen ingresadas las direcciones IPv4 e IPv6 que son insumos esenciales para el funcionamiento del servidor DNS en relación con respuestas autoritativas.

#### A. Diagrama de clases de la solución de software

En la figura 9 podrá encontrar de forma detallada las diferentes clases y métodos utilizados para poner en marcha el funcionamiento del servidor DNS.

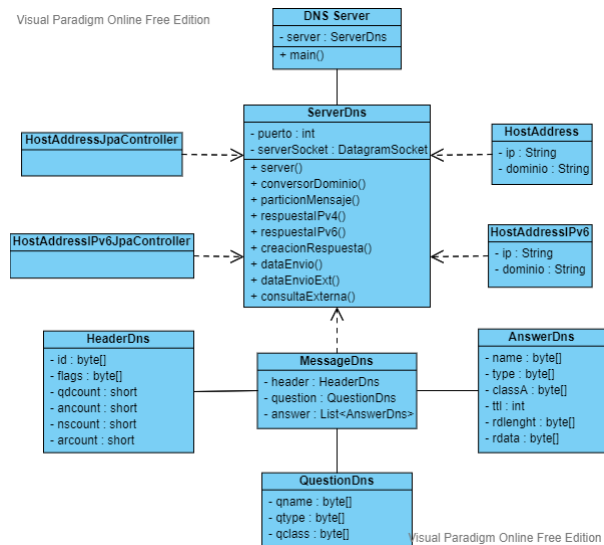


Fig. 9 Diagrama de clases

Cada clase tiene una funcionalidad específica en el sistema a operar, las cuales están definidas de la siguiente forma:

- DNS Server: Clase principal del aplicativo, permitirá instanciar el encendido del servidor DNS.
- ServerDns: Clase encargada de realizar todos los procesos para el buen funcionamiento del servidor DNS. La clase tiene los siguientes métodos:

TABLA I

Nombre Método	Descripción
server()	Instancia una sentencia while true que permitirá servir como un listener con el fin de que lleguen los datagramas de un cliente.
conversorDominio()	Traduce los bytes correspondientes al dominio recibido en el query a un String que puede ser procesado.
particionMensaje()	Realiza la división de los bytes recibidos en la query para que pueda ser instanciados y organizados como una clase MessageDns.
respuestaIPv4()	Realiza la gestión de la

	respuesta que se le dará al cliente si lo que necesita es una IPv4.
respuestaIPv6()	Realiza la gestión de la respuesta que se le dará al cliente si lo que necesita es una IPv6.
creacionRespuesta()	Clasifica la respuesta que se le dará al cliente, si la consulta es de tipo A instanciará el método respuestaIPv4() y si es AAAA instanciará respuestaIPv6().
dataEnvio()	Se encarga de dividir un objeto de clase MessageDns con el fin de obtener el arreglo de bytes que se enviarán al cliente.
dataEnvioExt()	Se encarga de dividir un objeto de clase MessageDns con el fin de obtener el arreglo de bytes que se enviarán al foreign resolver.
consultaExterna()	Se encarga de recibir enviar una instancia de MessageDns con el fin de que sea tomada como una query por parte del servidor externo, de igual forma, recibe la respuesta de dicho servidor para, posteriormente, enviarsela al cliente como una respuesta no autoritativa.

- MessageDns: Clase que tiene como funcionalidad ser el contenedor de los datos de un mensaje DNS. La clase internamente posee tres atributos, el header, question y una lista de answers.
- HeaderDns: Clase que representa el header de un mensaje DNS.
- QuestionDns: Clase que representa el apartado question de un mensaje DNS.
- HeaderDns: Clase que representa el apartado answer de un mensaje DNS.
- HostAddress: Clase creada a partir de la tabla en base de datos con el mismo nombre, representa las direcciones IPv4.
- HostAddressIPv6: Clase creada a partir de la tabla en base de datos con el mismo nombre, representa las direcciones IPv6.
- HostAddressJpaController: Clase creada con el fin de realizar consultas a la base de datos que almacena las direcciones IPv4 del servidor.
- HostAddressIPv6JpaController: Clase creada con el fin de realizar consultas a la base de datos que almacena las direcciones IPv6 del servidor.

## B. Diagrama de despliegue de la solución de software

En la figura 10 podrá encontrar el diagrama de despliegue que representa las interacciones entre elementos de la solución de software la cual representa al servidor DNS. En este se puede evidenciar los componentes para que el servidor DNS funcione correctamente.

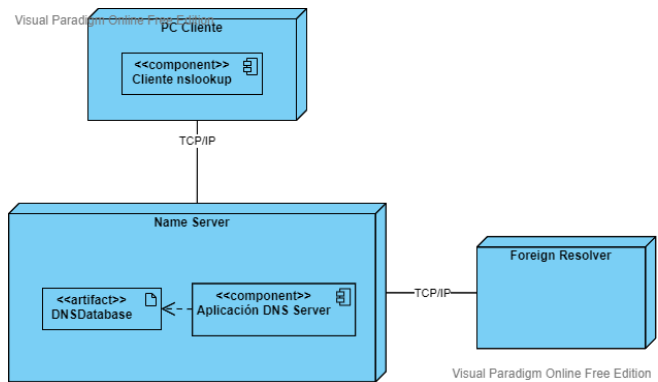


Fig. 10 Diagrama de despliegue

## IV. ESCENARIO DE PRUEBAS

El propósito de esta sección es definir todos los elementos que constituyen el escenario de pruebas construido con la herramienta emulador/simulador GNS3, que mediante sus configuraciones nos permitió asegurar la calidad del producto de software presentado en el trabajo del primer proyecto. A continuación se describirá más a detalle la topología usada y las diferentes configuraciones usadas en la misma.

### A. Descripción estructural de topología

La topología de red usada para comprobar los resultados del diseño e implementación del servidor DNS consta de dos routers configurados para que posean la funcionalidad del protocolo DHCP, con el fin de ofrecer ip 's dinámicas a los clientes que tienen asociados y el protocolo de enrutamiento dinámico RIP, esto con el fin de proveer a los routers rutas para cualquier dispositivo presente en la topología. Las redes de la topología se encuentran detalladas en la tabla 2.

TABLA II

IP de red	Descripción/Tipo	Máscara de subred
192.168.1.0	LAN	255.255.255.0 /24
192.168.2.0	LAN	255.255.255.0 /24
10.10.10.0	WAN	255.255.255.252 /30

Cabe resaltar que, para completar uno de los componentes de funcionalidad relacionados con las consultas hacia un servidor externo fue necesario hacer uso de una interface provista por medio del adaptador de red de VMware Network Adapter VMnet8. Su funcionalidad es principalmente servir como conexión indirecta al adaptador de red principal (en este caso por conexión ethernet) de la máquina de pruebas para establecer comunicación con la red de área local.

La topología de red está representada en la figura 11, en la cual se ve de forma detallada todos los componentes anteriormente mencionados.



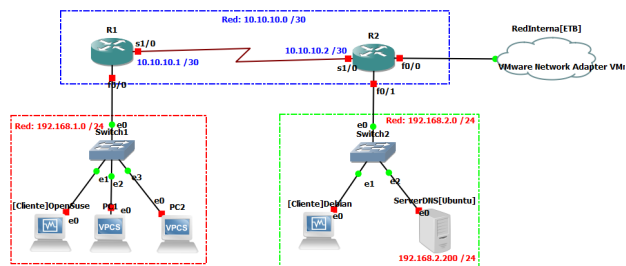


Fig. 11 Topología de red de escenario de pruebas

### B. Descripción de sistemas operativos utilizados para pruebas de cliente

Para el desarrollo del proyecto se utilizaron 3 distribuciones de Linux clasificadas y seleccionadas de acuerdo a su funcionamiento y desempeño. Estas fueron de vital importancia para el desarrollo de las pruebas ya que cada una cumple un papel específico según la topología diseñada en la anterior sección (ver fig. 3). También fueron buscadas las distribuciones menos pesadas según su tamaño y su correcto funcionamiento. Cada una de ellas fue descargada mediante su .iso para proceder con su instalación desde VirtualBox

¿Por que el uso de linux? Linux, es un sistema operativo basado en software libre con una amplia disponibilidad de distribuciones que permiten al usuario adaptarse según las necesidades y funciones que necesite. Además de ello, cualquier error presentado en su uso es posible resolverlo mayoritariamente con un mínimo esfuerzo con la puesta en marcha de comandos usados desde la terminal. Del mismo modo, mediante búsquedas de internet y gracias a la comunidad de software libre, no existe temor alguno y es muy poca la probabilidad de que haya un error fatal que no pueda ser resuelto. Por otro lado ofrece programas similares a los que son conocidos actualmente usados en sistemas operativos de software privativo como Windows o IOS como por ejemplo LibreOffice con funciones similares a Office 365 o GIMP un editor de texto similar a Photoshop. Otras ventajas que contiene es que es estable ya que siempre se van realizando actualizaciones en sus diferentes programas y distribuciones a parte de ser de alta calidad para servidores donde más del 90% son de este sistema operativo.

Las distribuciones de Linux usadas en nuestra topología son las siguientes:

- 1) Ubuntu (como servidor): Es una distribución de Linux que surgió en el año 2004 desarrollada por la empresa Canonical Ltd, fue basada en la distribución Debian y contiene el entorno de escritorio GNOME y actualmente se encuentra en la versión 7.04 que fue la utilizada en el proyecto. Incluye además la licencia GPL y está dividida en 4 servicios: Ubuntu Desktop, usado para áreas de trabajo personales desde un computador o máquina virtual, Ubuntu server que puede ser usado como un servidor de archivo simple o para guardar en un Cloud con 50 mil nodos, Ubuntu Cloud que se usa para OpenStack o creando su producción desde un Cluster, Ubuntu Flavours donde es posible escoger las configuraciones y aplicaciones que uno desea y Ubuntu for IoT para desarrolladores. El escogido por nuestro grupo fue Ubuntu Desktop donde se desarrolló todo el código presentado desde la aplicación Apache NetBeans. Su IP fue configurada manualmente establecido como 192.168.2.200 con una máscara de 24 [7].



Fig. 12 Logo de Ubuntu

- 2) Debian (como cliente de red interna): Es una distribución de Linux fundada en 1993 por Ian Murdock, patrocinada por el proyecto GNU y en sus inicios era la única abierta a desarrolladores y usuarios con oportunidades laborales. Actualmente se encuentra en la versión 10.9 que es la descargada en nuestro proyecto. Utiliza al igual que Ubuntu el entorno de escritorio GNOME además de Xfce, LXDE. Tiene licencias GPL, LGPL, BSD y más. Para su uso, fue necesario el comando nslookup que permite las consultas DNS hacia nuestro servidor con la implementación del protocolo UDP para la conexión. Su IP es configurada automáticamente por el DHCP integrado en el router al igual que su DNS, dirigiendola a nuestro servidor Ubuntu [8].



Fig. 13 Logo de Debian

- 3) openSUSE (cliente de red externa): Es una distribución de Linux lanzada en el año 2001, es basado como su nombre lo indica en OpenSource y a diferencia de SUSE Linux este es un proyecto libre. Tiene dos versiones, una reconocida con el nombre de Tumbleweed donde ofrece los últimos paquetes lanzados, un entorno de desarrollo integrado o una plataforma estable, ideal para cualquier tipo de usuarios. Su otra versión es el Leap que ofrece la distribución más útil y estable de las que lanza cada día, en el que se reciben actualizaciones y se puede configurar al gusto del cliente es ideal para desarrolladores, administradores y proveedores de software. Su versión actual es la 15.3 que fue la instalada en nuestro proyecto. Fue de gran importancia para

reconocer la amplitud de nuestro proyecto y que no está limitado a redes internas sino que host externos pueden también usar el servidor. Al igual que con la máquina virtual Debian, se usó el comando nslookup para simular el cliente DNS donde se realizan consultas a nuestro servidor. Su IP también es asignada por DHCP configurado en el router y su DNS como el servidor realizado por nosotros [9].

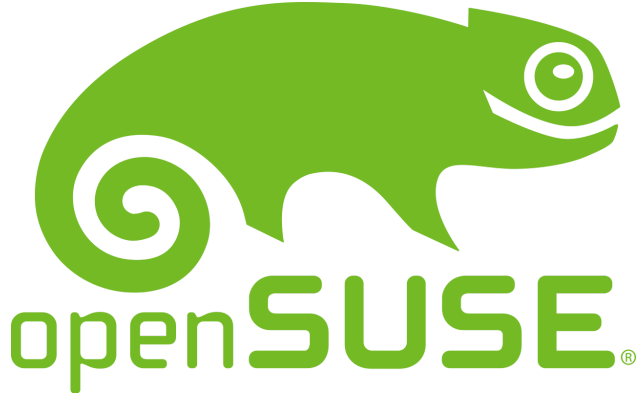


Fig. 14 Logo de openSuse

## V. PROTOCOLO Y DISEÑO DE PRUEBAS Y RESULTADOS

### 1) Primera prueba

Se realizó el código que describe el protocolo udp desde la máquina virtual Ubuntu como servidor y la máquina real como cliente. Para esta primera prueba desarrollamos además el código del protocolo UDP del cliente simplemente para analizar su correcto funcionamiento el cual acordamos eliminarlo en las siguientes pruebas luego de implementar en nuestra topología las máquinas que servirán como cliente. Por consiguiente, activamos nuestro programa desde el main de nuestro servidor donde se encontraba el código descrito anteriormente y se probó enviando una cadena desde el cliente que es nuestra máquina real para confirmar que el protocolo estaba funcionando correctamente. La respuesta fue efectiva ya que se logró presentar el mensaje en el servidor y al igual recibió respuesta por parte del mismo.

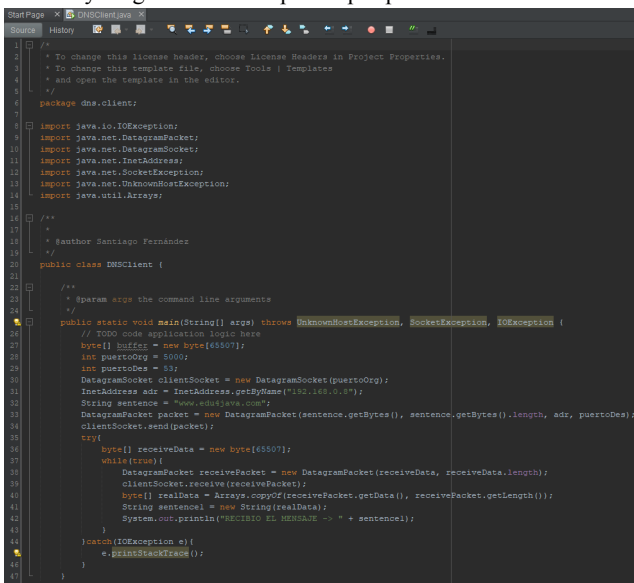


Fig. 15 Código del cliente UDP

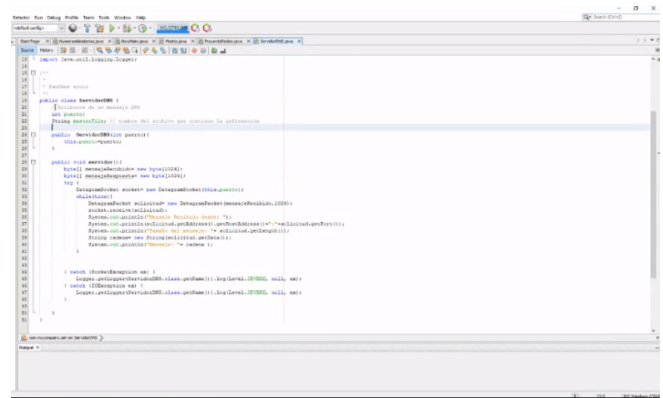


Fig. 16 Código del servidor UDP

### 2) Segunda prueba

Se procedió a realizar la construcción de la topología, esta vez poniendo de cliente a nuestra máquina virtual Debian y eliminando el código del cliente UDP que teníamos en nuestra máquina real, para así hacer uso del comando lookup para mandar la consulta DNS, sin embargo no se recibió ningún paquete en nuestro servidor.

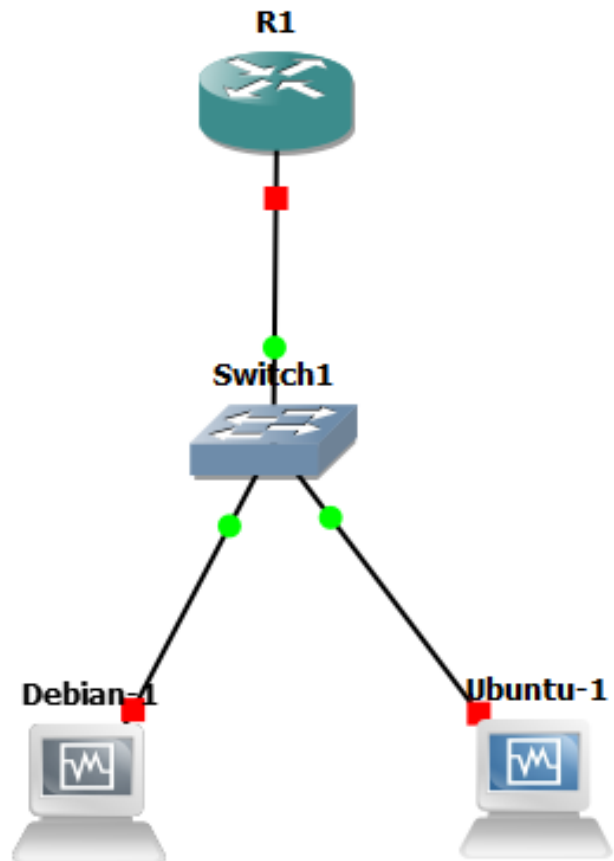
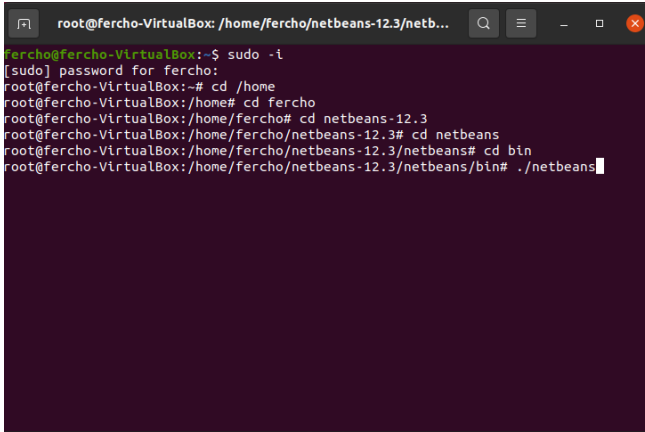


Fig. 17 Adición de la VM Debian como cliente

### 3) Tercera prueba

Luego de lo sucedido en la prueba anterior, encontramos que el puerto que habíamos colocado no era el indicado, así que fue cambiado por el 53, no obstante, tuvimos que hacer uso del comando sudo en la máquina virtual para poder entrar como usuario root y tener todos los permisos necesarios, consiguiendo esta vez una respuesta efectiva donde imprimimos cada elemento que contiene el formato del mensaje DNS.

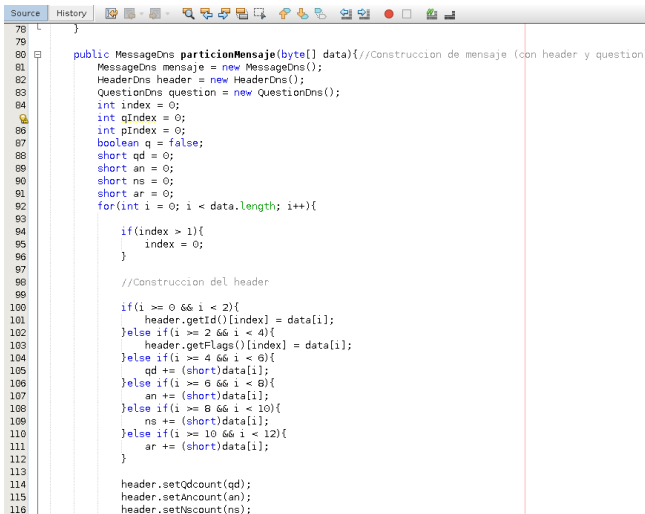


```
root@fercho-VirtualBox: /home/fercho/netbeans-12.3/netb...
fercho@fercho-VirtualBox:~$ sudo -i
[sudo] password for fercho:
root@fercho-VirtualBox:~# cd /home
root@fercho-VirtualBox:~/home# cd fercho
root@fercho-VirtualBox:~/home/fercho# cd netbeans-12.3
root@fercho-VirtualBox:~/home/fercho/netbeans-12.3# cd netbeans
root@fercho-VirtualBox:~/home/fercho/netbeans-12.3/netbeans# cd bin
root@fercho-VirtualBox:~/home/fercho/netbeans-12.3/netbeans/bin# ./netbeans
```

Fig. 18 Comando root para permitir acceso al puerto

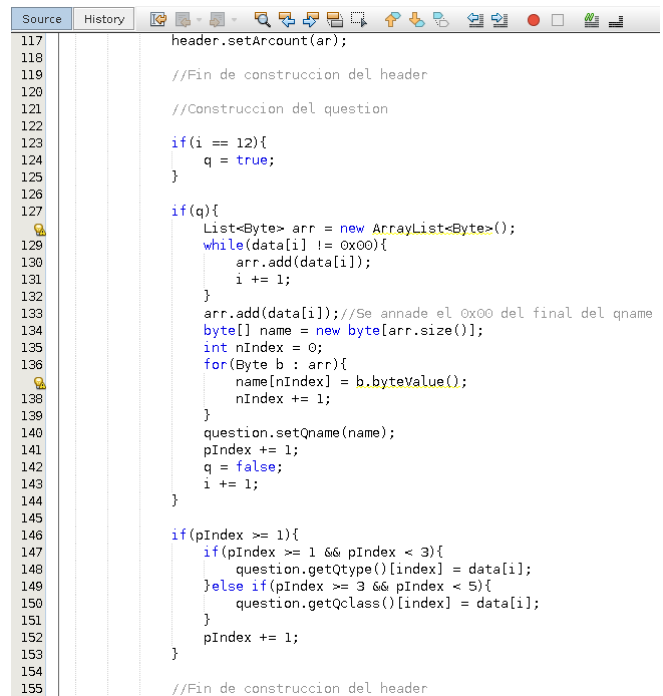
### 4) Cuarta prueba

En esta prueba pasamos el código de nuestro main a la función server(), procedimos además a implementar la función partición mensaje para tener mayor claridad de cada elemento encontrado en el paquete. Para ello, fue necesario crear la nueva clase HeaderDns donde cada atributo se acordó establecerlo como arreglo de bytes, al igual que los atributos de la clase QuestionDns, ambos recopilados en la clase MessageDns, se hizo la prueba de su correcto funcionamiento y efectivamente se guardó cada elemento en nuestra clase.



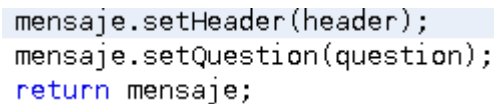
```
78
79
80 public MessageDns particiónMensaje(byte[] data){//Construccion de mensaje (con header y question)
81     MessageDns mensaje = new MessageDns();
82     HeaderDns header = new HeaderDns();
83     QuestionDns question = new QuestionDns();
84     int index = 0;
85     int qIndex = 0;
86     int pIndex = 0;
87     boolean q = false;
88     short qd = 0;
89     short an = 0;
90     short ns = 0;
91     short ar = 0;
92     for(int i = 0; i < data.length; i++){
93
94         if(index > 1){
95             index = 0;
96         }
97
98         //Construccion del header
99
100         if(i >= 0 && i < 2){
101             header.getId() [index] = data[i];
102         }else if(i >= 2 && i < 4){
103             header.getFlags() [index] = data[i];
104         }else if(i >= 4 && i < 6){
105             qd += (short) data[i];
106         }else if(i >= 6 && i < 8){
107             an += (short) data[i];
108         }else if(i >= 8 && i < 10){
109             ns += (short) data[i];
110         }else if(i >= 10 && i < 12){
111             ar += (short) data[i];
112         }
113
114         header.setQdcount(qd);
115         header.setAncount(an);
116         header.setNscount(ns);
117     }
118 }
```

Fig. 19 Construcción del header en nuestro servidor



```
117 header.setArcount(ar);
118
119 //Fin de construccion del header
120
121 //Construccion del question
122
123 if(i == 12){
124     q = true;
125 }
126
127 if(q){
128     List<Byte> arr = new ArrayList<Byte>();
129     while(data[i] != 0x00){
130         arr.add(data[i]);
131         i += 1;
132     }
133     arr.add(data[i]); //Se anade el 0x00 del final del qname
134     byte[] name = new byte[arr.size()];
135     int nIndex = 0;
136     for(Byte b : arr){
137         name[nIndex] = b.byteValue();
138         nIndex += 1;
139     }
140     question.setQname(name);
141     pIndex += 1;
142     q = false;
143     i += 1;
144 }
145
146 if(pIndex >= 1){
147     if(pIndex >= 1 && pIndex < 3){
148         question.getQtype() [index] = data[i];
149     }else if(pIndex >= 3 && pIndex < 5){
150         question.getQclass() [index] = data[i];
151     }
152     pIndex += 1;
153 }
154
155 //Fin de construccion del header
```

Fig. 20 Construcción del question en nuestro servidor

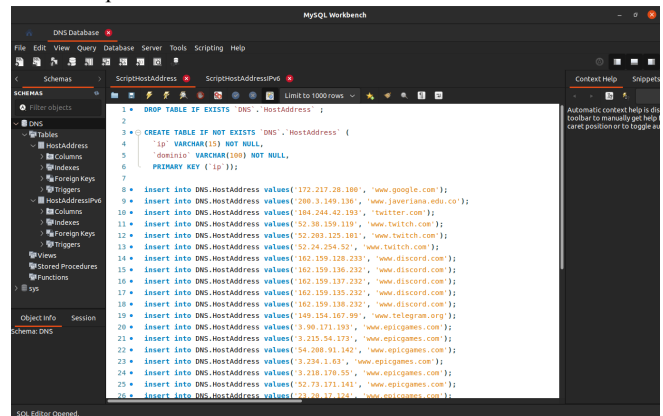


```
mensaje.setHeader(header);
mensaje.setQuestion(question);
return mensaje;
```

Fig. 21 Incorporación del header y question a nuestro mensaje.

### 5) Quinta prueba

Se procedió con la creación de la respuesta, el cual se usó la base de datos MySQL donde se guardaron todas nuestras direcciones del nameserver, de ahí hubo complicaciones de incluirla en la aplicación NetBeans por la versión instalada, sin embargo se pudo implementar mediante la instalación de la librería jdbc. Luego de ello, se añadió cada elemento del mensaje de acuerdo a la query recibida por el usuario, la lista de direcciones encontradas según el dominio y los elementos QR y AA fueron colocados como 1, haciendo saber que este mensaje es un response y su respuesta es autoritativa. Al finalizar la incorporación de los elementos del mensaje, se realizó la prueba desde el cliente y se verificó la veracidad del mensaje, el cual obtuvo una respuesta negativa por el hecho de que no se formó correctamente el formato del DNS.



```
1 DROP TABLE IF EXISTS `DNS`.`HostAddress` ;
2
3 CREATE TABLE IF NOT EXISTS `DNS`.`HostAddress` (
4   `ip` VARCHAR(15) NOT NULL,
5   `dominio` VARCHAR(100) NOT NULL,
6   PRIMARY KEY (`ip`) );
7
8 Insert into DNS.HostAddress values('172.217.28.108','www.google.com');
9 Insert into DNS.HostAddress values('208.3.149.130','www.javerlana.edu.co');
10 Insert into DNS.HostAddress values('184.244.42.193','twitter.com');
11 Insert into DNS.HostAddress values('52.58.159.119','www.twitch.com');
12 Insert into DNS.HostAddress values('52.283.225.181','www.twitch.com');
13 Insert into DNS.HostAddress values('52.24.254.52','www.twitch.com');
14 Insert into DNS.HostAddress values('162.159.128.233','www.discord.com');
15 Insert into DNS.HostAddress values('162.159.136.232','www.discord.com');
16 Insert into DNS.HostAddress values('162.159.137.232','www.discord.com');
17 Insert into DNS.HostAddress values('162.159.135.232','www.discord.com');
18 Insert into DNS.HostAddress values('162.159.138.232','www.discord.com');
19 Insert into DNS.HostAddress values('149.149.167.99','www.telegram.org');
20 Insert into DNS.HostAddress values('3.90.171.193','www.epicgames.com');
21 Insert into DNS.HostAddress values('3.215.54.173','www.epicgames.com');
22 Insert into DNS.HostAddress values('54.208.91.142','www.epicgames.com');
23 Insert into DNS.HostAddress values('3.238.1.43','www.epicgames.com');
24 Insert into DNS.HostAddress values('3.218.178.55','www.epicgames.com');
25 Insert into DNS.HostAddress values('52.79.171.141','www.epicgames.com');
```

Fig. 22 Creación base de datos en MySQL

## 6) Sexta prueba

Según el resultado de la quinta prueba, se analizaron los motivos del mensaje mal formado y se encontró que al pasar cada elemento nuestra clase a bytes había un error en la implementación así que procedimos en cambiar los tipos de datos bytes[] a enteros, booleanos o short según como el RFC 1035 lo planteaba. Así que se hizo la prueba y terminó resultando una respuesta positiva a las mejoras que planteamos en nuestro código logrando así tener la respuesta correcta del DNS query que el cliente había solicitado.

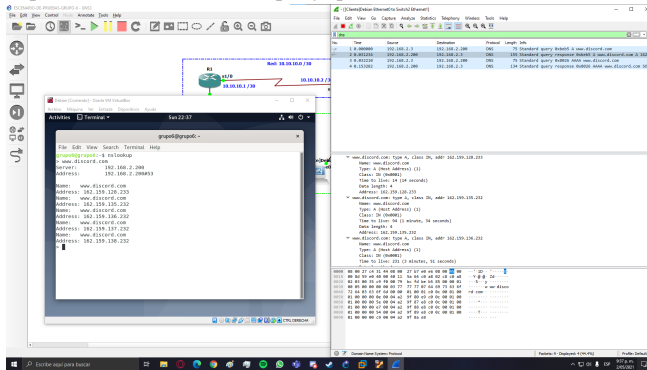


Fig. 23 Resultado de una consulta IPv4

## 7) Séptima prueba

Luego de acertar en los DNS response que ofrece nuestro name server, se procedió a hacer uso del foreign resolver para las consultas que no podían ser resueltas, para ello, se implementó en el código de nuestro programa la función consulta externa el cual su misión es lograr la conexión con el servidor DNS de Google (8.8.8.8) y en este caso, ser el cliente del protocolo para poder recibir la respuesta de las direcciones que no pudieron ser encontradas de acuerdo al dominio acogido de la query realizada. De allí, pudimos notar la necesidad de tener una conexión a internet en nuestra máquina virtual. Así que se llevó a cabo el uso del Cloud en GNS3 dentro de nuestra topología que permitiera la conexión que tiene nuestra máquina real con la red Ethernet. Por tal motivo, se hicieron las configuraciones respectivas desde los router y no se obtuvo respuesta alguna.

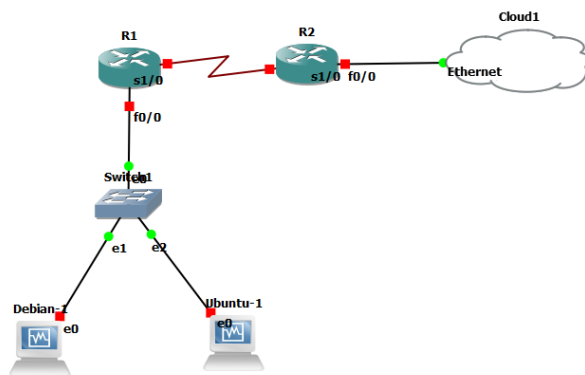


Fig. 24 Intento conexión de red desde la topología

## 8) Octava prueba

Al carecer de una respuesta asertiva en la prueba anterior se llegó al acuerdo de implementar las consultas IPv6. Así que la función de respuesta diseñada anteriormente se usó para clasificar las respuestas en IPv4 e IPv6 de acuerdo con su tipo y se añadió una nueva tabla en nuestra base de datos donde almacenamos las direcciones y sus dominios que aplicaran a este tipo de consultas.

Finalmente se logró obtener la respuesta de manera inmediata y no hubo mayor complicación en añadir este tipo de consultas.

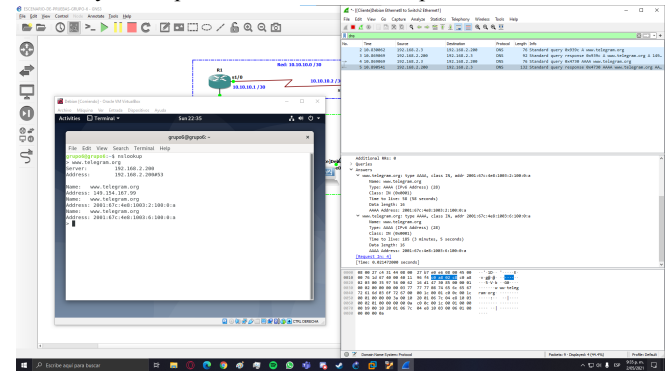


Fig. 25 Resultado de una consulta IPv6

## 9) Novena prueba

Finalmente, luego de desarrollar toda la implementación de las consultas hacia nuestro name server, se consultó por varios medios sobre el error sucedido en la séptima prueba y fue posible la resolución por medio de la aplicación VMware. Gracias a esta, fue posible el uso de internet en nuestro servidor y permitir correctamente la solución de las consultas establecidas por el cliente desde el foreign resolver.

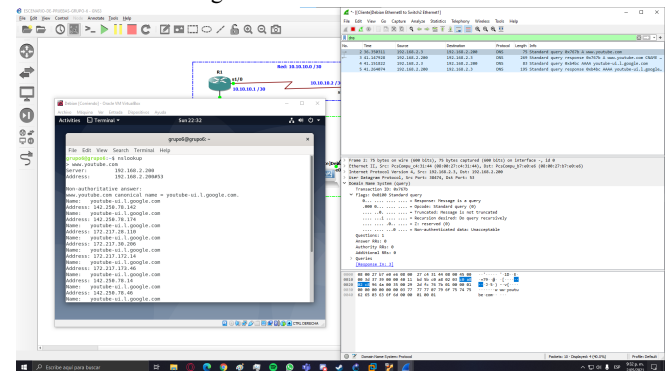


Fig. 26 Resultado haciendo uso del Foreign Resolver desde el cliente

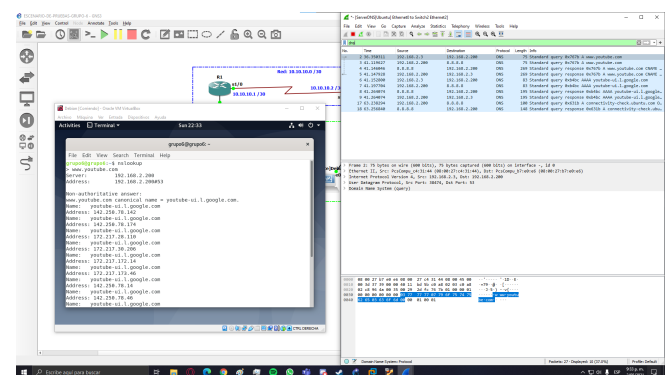


Fig. 27 Resultado haciendo uso del Foreign Resolver desde el servidor

## VI. CONCLUSIONES

1. El protocolo DNS es muy usado en navegadores cuando se desea ingresar a un dominio y no se conoce su IP, permitiendo dar de forma óptima las comunicaciones de hoy en día..
2. Wireshark es una herramienta muy útil que permite visualizar el intercambio de paquetes, además de proveer la información detallada de cada uno.



3. La aplicación GNS3 es de gran ayuda para simular un diagrama de red con capacidad de configuraciones de modelos reales físicos.

#### REFERENCIAS

- [1] «IBM Docs», abr. 14, 2021.  
<https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/es/i/7.2?topic=services-domain-name-system> (accedido may 02, 2021).
- [2] C. S. González, «Análisis de vulnerabilidades del DNS», p. 58.
- [3] A. S. Tanenbaum, «Redes de Computadoras», p. 914.
- [4] «Capa de aplicación (Guía de administración del sistema: servicios IP)».  
<https://docs.oracle.com/cd/E19957-01/820-2981/ipov-22/index.html> (accedido may 02, 2021).
- [5] J. Postel, «User Datagram Protocol».  
<https://tools.ietf.org/html/rfc768> (accedido may 02, 2021).
- [6] P. V. Mockapetris, «Domain names - implementation and specification». <https://tools.ietf.org/html/rfc1035> (accedido may 02, 2021).
- [7] «Historia de Ubuntu».  
<https://www.ubuntu.mx.org/historia.php> (accedido may 02, 2021).
- [8] «Una breve historia de Debian», p. 27.
- [9] C. S. front-end and U. design Zvezdana Marjanovic: graphic, «The makers' choice for sysadmins, developers and desktop users.», *openSUSE*. <https://www.opensuse.org> (accedido may 02, 2021).