Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

Implementación de un Demonio LLMNR Para el Sistema Operativo GNU/Linux

Trabajo Especial de Grado
Presentado ante la ilustre
Universidad Central de Venezuela
Por el bachiller:

Julio C. Garroz Riveros
V-17.115.364
garroz.jc@gmail.com

Tutores: Prof. Eric Gamess y Prof. Roger Bello

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

ACTA DE VEREDICTO

Quienes suscriben, miembros del jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por el Bachiller Julio Garroz C.I. V-17.115.364, con el título "**Implementación de un Demonio LLMNR para el Sistema Operativo GNU/Linux**", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el nombrado trabajo por cada uno de los miembros del jurado, éste fijó el día jueves 11 de mayo de 2017, para que su autor lo defendiera en forma pública, en la Sala de Internet II, Laboratorio LACORE, de la Escuela de Computación, mediante una exposición oral de su contenido, luego de la cual respondió satisfactoriamente a las preguntas que le fueron formuladas por el jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela.

Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

Es de aclarar que el Profesor Eric Gamess se encuentra de permiso fuera del país. Por esta razón, estuvo via videoconferencia, de común acuerdo con los demás miembros del jurado. Por ende, el Profesor Robinson Rivas, director de la Escuela de Computación, firma el presente documento en su lugar.

En fe de lo cual se levanta el presente acta en Caracas a los 11 días del mes de mayo del año 2017, dejando constancia que el Profesor Roger Bello actuó como coordinador del jurado.

REPUBLICA DE VENEZUELA
Facultad de Ciencias
Escuela de Computación
Universidad Central - Caracas

Prof. Eric Gamess
Tutor

Prof. Roger Bello
Tutor

Prof. Antonio Russoniello
Jurado Principal

Prof. Dedaniel Urribarri
Jurado Principal

Agradecimientos

Quiero agradecer en este Trabajo Especial de Grado a mis padres, mis hermanos y todos los familiares que siempre me brindaron consejo y apoyo.

Agradezco especialmente a mi padre, Guillermo Garroz, por tanto cariño, paciencia, apoyo, comprensión, por siempre querer y buscar lo mejor para mí.

A mi tutor Eric Gamess, por todo su apoyo y paciencia, por ser excelente persona y educador.

A Roger y Antonio, y toda la gente que hace vida en LACORE.

Por ultimo quiero agradecer a todos los amigos y compañeros que compartieron conmigo a lo largo de la carrera, especialmente a P.P., por dar tanto y recibir tan poco.

Julio Garroz

Resumen

Título:
Implementación de un Demonio LLMNR para el Sistema Operativo GNU/Linux.

Autor:
Julio Garroz

Tutores:
Prof. Eric Gamess y Profesor Roger Bello.

El presente Trabajo Especial de Grado está enfocado en el desarrollo de un Demonio (Disk and Execution Monitor) para el protocolo LLMNR que cumpla con las especificaciones descritas en el RFC 4795, que corra sobra el Sistema Operativo GNU/Linux y que funcione sobre los protocolos IPv4 e IPv6. Con este Demonio se busca ofrecer una solución que permita llenar el vacío existente al respecto, dado que hasta el día de hoy, en los repositorios oficiales de las principales distribuciones de GNU/Linux no existe ninguna implementación para dicho protocolo. LLMNR es un protocolo de red que busca realizar una única y sencilla tarea: Resolver nombres de equipo y/o dominios en una red LAN, sin requerir ningún tipo de configuración por parte del usuario y/o administrador del sistema.

El primer paso para el desarrollo de este trabajo fue la recopilación de información que pudiera ser de interés para el desarrollo del demonio. Con este fin se realizó un estudio del comportamiento de la implementación del protocolo LLMNR existente en el Sistema Operativo Windows 7. Posterior a eso se realizó el estudio sobre cómo implementar un demonio sobre el Sistema Operativo GNU/Linux y se concluyó, de manera inequívoca, que el lenguaje de programación a utilizar debía ser C, junto con el compilador GCC y la herramienta *make* para compilar.

Finalmente se realizó de manera satisfactoria el desarrollo de un demonio LLMNR, que cumple con los requisitos especificados en el RFC 4795, perfectamente integrado, tanto como Sender y Responder, sobre el Sistema Operativo GNU/Linux, el cual puede configurarse (si el usuario y/o administrador así lo requiere más sin embargo no es requisito para operar) a través de un archivo de configuración.

Palabras claves: LLMNR, Linux, Daemon, C, DNS.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

1. **Introducción**

DNS (Domain Name System), descrito en el RFC 1034 [**1**] y el RFC 1035 [**2**], es un protocolo diseñado para la resolucion de nombres de alto nivel. Permite mapear un nombre de alto nivel a una dirección IP y viceversa. Con este propósito ha sido usado a través de Internet y emplea un conjunto de servidores, geográficamente distribuidos, para realizar dicha tarea.

LLMNR (Link-Local Multicast Name Resolution), descrito en el RFC 4795 [**3**], es un protocolo basado en DNS que provee un método de resolución de nombres de equipos que se encuentran en una LAN (Local Area Network), la cual no posee un servidor DNS. Está basado en el mismo formato de paquete DNS, sin embargo, LLMNR no pretende ser un sustituto del mismo, ya que opera a nivel de redes LAN.

El objetivo de la investigación es presentar LLMNR e indagar acerca de los distintos aspectos que deben ser tomados en cuenta para la creación de un demonio LLMNR para las plataformas basadas en GNU/Linux. Un demonio o DAEMON (Disk And Execution Monitor) es un proceso que se ejecuta en segundo plano, esperando un evento o condición para realizar una tarea específica.

El resto del documento se encuentra estructurado de la siguiente forma:

Capítulo 2. El Problema
En este capítulo se presenta la descripción del problema así como los objetivos, generales y específicos.

Capítulo 3. IPv6 (Internet Protocol versión 6)
En este capítulo se da una introducción a Internet Protocol versión 6 (IPv6) y se describen las características más importantes de este protocolo así como su estructura básica.

Capítulo 4. Descripción del Protocolo LLMNR
En este capítulo se describen las características del protocolo LLMNR, su método de resolución de conflictos, y los problemas de seguridad inherentes.

Capítulo 5. Trabajos Relacionados
En este capítulo, se presentan los protocolos relacionados como mDNS (Multicast DNS), el cual es una propuesta realizada por Apple, que cumple una función similar a LLMNR. También se habla sobre protocolos “Zero Configuration”.

Capítulo 6. Marco Metodológico
En este capítulo se describe el marco metodológico

Capítulo 7: Marco Aplicativo
En este capítulo se describe, basado en el marco metodológico, la implementación del demonio o aplicación, y los distintos aspectos que lo componen.

Capítulo 8: Pruebas y Análisis de Resultados
En este capítulo se realizan pruebas de funcionamiento, interoperabilidad, rendimiento y estrés sobre el demonio desarrollado.

Capítulo 9: Conclusiones y Trabajos Futuros
Contiene las conclusiones de este trabajo especial de grado y una sugerencia o recomendación sobre posibles trabajos futuros.

2. El Problema

2.1 Planteamiento del Problema

La resolución de nombres es el proceso mediante el cual se transforma un nombre a una (o varias) direcciones IP. Tradicionalmente este proceso se ha venido realizando (de manera exitosa) a través del sistema DNS, particularmente este ha sido exitoso en Internet, pero cuando de redes locales (LANs) se trata el sistema DNS presenta ciertos inconvenientes, como el costo y los conocimientos técnicos requeridos para implementar un servidor DNS. Análogamente, para las soluciones alternas al sistema DNS (algunas de ellas mencionadas en el Capítulo 0), se presentan problemas similares a la hora de implementar, como en el caso de NetBIOS (Network Basic Input Output System) y WINS (Windows Internet Name Service), más el añadido que requeriría ciertos conocimientos técnicos (extras) si se quisiera lograr la integración de las soluciones antes mencionadas con el sistema DNS, por lo que surge la necesidad de implementar una solución que no adolezca de los problemas previamente planteados.

En la actualidad el único protocolo existente para sistemas basados en GNU/Linux que tenga como propósito resolver nombres de equipo y/o dominio dentro de una red local es multicast DNS (mDNS). Sin embargo, para mDNS existen 2 limitantes importantes:

- Solo resuelve nombres con sufijos ".local"
- No es compatible con el Sistema Operativo Microsoft Windows puesto que este último no soporta mDNS.

Debido a esto,2 máquinas con sistemas operativos distintos (GNU/Linux y Windows) no podrían comunicarse en caso de que alguna de las 2 quisiera resolver el nombre de equipo y/o dominio de la otra (podrían hacerlo a través de NetBIOS pero este protocolo se encuentra "deprecated", es decir, en periodo de extinción).Tomando en cuenta todo esto se desarrolló una implementación del protocolo LLMNR para el sistema operativo GNU/Linux, el cual cumple con los requisitos especificados en el RFC 4795 [**3**] y que ofrece buen soporte para configurar cualquier parámetro que sea necesario.

2.2 Justificación

El uso de las redes informáticas ha tenido un crecimiento exponencial en las ultimas 2 décadas. Han revolucionado la manera en que las personas trabajan, influyendo de manera directa en el aumento de la eficiencia y productividad, hasta lograr convertirse en una parte fundamental de empresas, gobiernos, institutos, organizaciones y particulares. Esto, aunado a la aparición (y eventual dominio sobre las redes informáticas) del protocolo IPv6 (que tiene como característica la simplificación del proceso de configuración) y a la falta de una implementación (para GNU/Linux) de un protocolo de resolución de nombres que cumpla con las características deseadas, surge la necesidad de realizar una implementación de LLMNR que cumpla con el estándar descrito en [**3**]

2.3 Objetivos

2.3.1 Objetivo General

Diseñar e implementar un demonio LLMNR para sistemas operativos basados en GNU/Linux

2.3.2 Objetivos Específicos

- Estudiar el protocolo LLMNR definido en el RFC 4795 [**3**].
- Estudiar protocolos similares como WINS, mDNS y otros.
- Estudiar y solidificar los conocimientos sobre programación de sockets.
- Desarrollar un demonio LLMNR (*llmnrd*) basado en el RFC 4795 [**3**].
- Desarrollar una herramienta que permita realizar pruebas de rendimiento y estrés sobre el demonio propuesto (*llmnrd*) y *ms-llmnr* [**4**].
- Realizar pruebas de interoperabilidad entre el demonio LLMNR desarrollado y *ms-llmnr* [**4**].
- Realizar pruebas de rendimiento sobre el demonio *llmnrd* y *ms-llmnr*.

2.4 Alcances

El demonio desarrollado debe ser capaz de actuar como LLMNR Sender y Responder, sobre los protocolos IPv4 e IPv6, altamente fiable, de fácil instalación y configuración sobre los sistemas GNU/Linux y capaz de interactuar con su contraparte *ms-llmnr* [**4**] sin ningún tipo de problema.

3. **Internet Protocol versión 6 (IPv6)**

IPv4 es un protocolo que tiene más de veinte años y que ha probado ser un protocolo bastante robusto, fácil de implementar e interoperable. Ha logrado exitosamente escalar de una red local a una global como lo es Internet, propósito para el cual no fue diseñado originalmente. Debido a esto, en su diseño original no era posible anticipar lo siguiente:

- Crecimiento exponencial de Internet y agotamiento del espacio de direcciones IPv4.
- La necesidad de una configuración más simple y automatizada.
- Requerimientos de seguridad a nivel de la capa de Internet (capa 3).
- La necesidad de un mejor soporte para datos priorizados o de tiempo real.

Para solventar estas y otras necesidades el "Internet Engineering Task Force" (IETF) desarrolló un conjunto de protocolos conocido como IPv6, el cual incorpora conceptos y métodos para realizar la actualización requerida en IPv4. IPv6 fue diseñado para tener un mínimo impacto en las capas de protocolos (tanto inferior como superior), así como evitar la necesidad de agregación dinámica de características en el protocolo (como ocurrió con IPv4).

3.1 Características

A continuación se enumeran las principales características de IPv6:

- **Direccionamiento extendido:** IPv6 implementa direcciones de 128 bits permitiendo tener $3.4\text{x } 10^{38}$combinaciones posibles de direcciones. Adicionalmente el espacio de direcciones de IPv6 se diseñó para permitir diferentes niveles de *subnetting* y asignación de direcciones.
- **Nuevo formato de la cabecera**: Aun cuando el encabezado aumenta de tamaño, se ha simplificado el mismo con el fin de minimizar el procesamiento y pasa de tener 13 campos (IPv4) a tener 8 campos (IPv6).
- **Soporte para opciones mejorado**: En IPv4 el espacio para opciones va de 0 a 40 bytes, mientras que en IPv6 se maneja con encabezados de extensión, lo cual permite encadenar las opciones y no tener un límite de espacio.
- **Direccionamiento jerárquico**: Las direcciones IPv6 (o grupo de direcciones IPv6) son asignadas de manera jerárquica por un RIR (Regional Internet Registry).
- **Direcciones multicast mejoradas**: Se eliminaron las direcciones de broadcast. Si es necesario enviar un mensaje a todos los nodos de una misma subred, se hace a través de una dirección multicast. Este mecanismo ya existía en IPv4 pero ahora toma fuerza y las direcciones multicast de IPv6 tienen un alcance.
- **Direcciones anycast**: IPv6 introduce el concepto de direcciones anycast, las cuales sirven para enviar un paquete al nodo más cercano de un grupo determinado, en base a las métricas definidas por el protocolo de enrutamiento.
- **Mejor soporte para tráfico priorizado:** Nuevos campos en la cabecera IPv6 definen como el tráfico es identificado y manejado, proveyendo un mejor soporte para calidad de servicio.
- **Fragmentación en el emisor**: En IPv4 la fragmentación se puede dar tanto en el emisor como en cualquier dispositivo intermedio de la capa de red. En IPv6, sólo ocurre en el emisor lo cual disminuye la carga de los routers.
- **Autoconfiguración**: Otro aspecto muy importante de IPv6 es la autoconfiguración, que permite que los dispositivos finales de una red obtengan automáticamente sus direcciones IPv6.

- **Soporte requerido para IPsec:** El soporte para las cabeceras de IPsec es requerido en las implementaciones de IPv6 con el fin de tener mecanismos de seguridad en la capa 3.
- **Nuevo protocolo para la interacción entre nodos vecinos:** El protocolo "Neighbor Discovery" es una serie de mensajes ICMPv6 (Internet Control Message Protocol version 6) que maneja la interacción entre nodos vecinos. Neighbor Discovery reemplaza y mejora el uso de los protocolos ARP e ICMPv4.

3.2 Cabecera IPv6

La Figura 3.1 (tomada de [**5**]) muestra la cabecera IPv6. Sus diferentes campos son presentados a continuación.

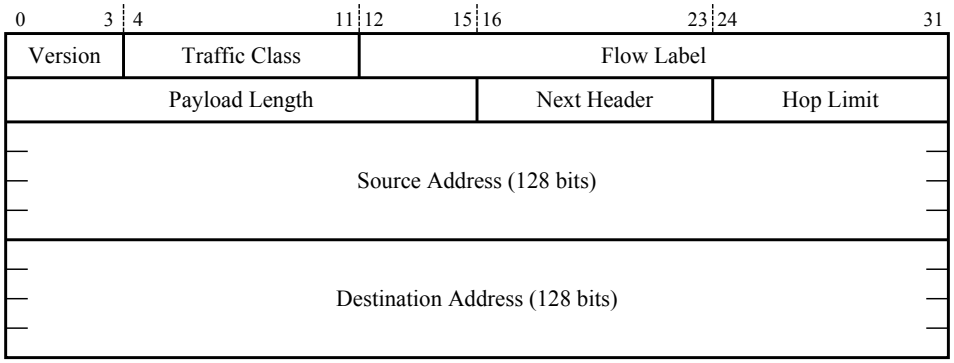


Figura 3.1: Cabecera IPv6

- **Version**: El tamaño de este campo es de 4 bits. Indica la versión del protocolo de Internet que se utiliza. Para IPv6 el valor del campo siempre debe ser 6.
- **Traffic Class**: El tamaño de este campo es de 8 bits. Indica la clase del paquete IPv6 o su prioridad. Los nodos emisores y routers intermedios pueden utilizarlo para identificar y distinguir entre diferentes clases o prioridades de paquetes IPv6. Este campo es equivalente al campo TOS (Type of Service) de IPv4.
- **Flow Label**: El tamaño de este campo es de 20 bits. Indica que un paquete IPv6 pertenece a una secuencia específica de paquetes entre un origen y un destino y es usado para el manejo de tráfico priorizado.
- **Payload Length**: El tamaño de este campo es de 16 bits. Indica la longitud en bytes de la carga útil, incluyendo las cabeceras de extensión. El valor debe ser un entero sin signo.
- **Next Header**: El tamaño de este campo es de 8 bits. Contiene un número que identifica el protocolo o cabecera de extensión que sigue inmediatamente después de la cabecera IPv6.
- **Hop Limit**: El tamaño de este campo es de 8 bits. Especifica el número máximo de routers por los cuales puede ser transmitido un paquete antes de ser descartado.
- **Source Address**: El tamaño de este campo es de 128 bits. Especifica la dirección IPv6 del origen de un mensaje.
- **Destination Address**: El tamaño de este campo es de 128 bits. Especifica la dirección IPv6 del receptor del mensaje. Si la cabecera de extensión "Routing Header" está presente, es posible que la dirección de destino no sea el nodo final sino algún nodo intermedio.

3.2.1 Cabeceras de Extensión

En la cabecera IPv4, existe un campo para las opciones que es usado para definir información adicional sobre el paquete o sobre la manera en que este debería ser

procesado. Todos los routers (al menos que se indique lo contrario) deben procesar las opciones de la cabecera de IPv4. Dicho procesamiento puede mermar en el desempeño del router [**6**]. Para el diseño de IPv6 se tomó en cuenta el impacto en desempeño que conlleva el procesamiento de las opciones de IPv4. Es por esto que en IPv6 fueron removidas de la cabecera y colocadas como cabeceras de extensión, las cuales son encadenadas al final de la cabecera principal de IPv6, solo cuando son requeridas. En la Figura 3.2 se puede apreciar un paquete IPv6 que contiene cabeceras de extensión.

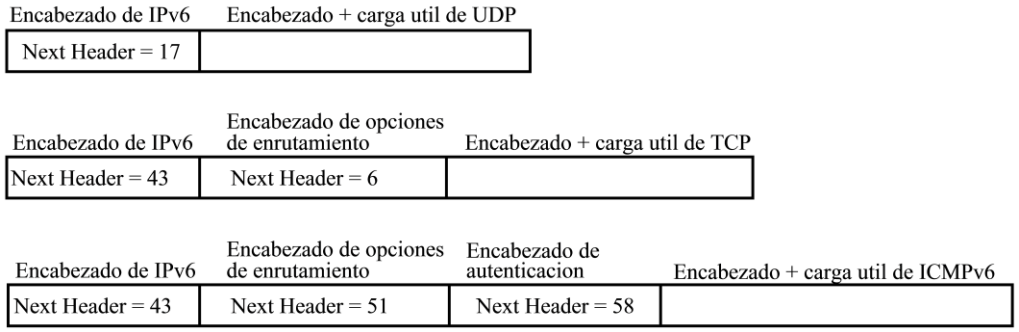


Figura 3.2: Paquete IPv6 con Encadenamiento de Cabeceras de Extensión

Existen seis cabeceras de extensión distintas definidas en [**5**] y se listan a continuación en la Tabla 3.1junto con el valor de “Next Header” asignado:

Order	Header Type	Next Header Code
1	Basic IPv6 Header	-
2	Hop-by-Hop Options	0
3	Destination Options	60
4	Routing Header	43
5	Fragment Header	44
6	Authentication Header	51
7	Encapsulation Security Payload Header	50
8	Mobility Header	135
9	No Next Header	59

Tabla 3.1: Cabeceras de Extensión IPv6

Un paquete IPv6 puede contener cero o más cabeceras de extensión localizadas entre la cabecera IPv6 y la cabecera del protocolo de capa superior (TCP, UDP, ICMPv6, etcétera.). La longitud de las cabeceras de extensión debe ser un número entero múltiplo de 8 bytes. En caso de que la opción no cumpla con dicho requerimiento se utilizarán bytes de relleno para lograrlo.

Las cabeceras de extensión son procesadas en el orden en que se presentan. Los nodos intermedios sólo toman en cuenta la opción “Hop-by-Hop” por lo tanto ésta debe ser la primera cabecera de extensión.

3.3 Direccionamiento en IPv6

Los aspectos relacionados con las direcciones IPv6, tales como su representación y sus diferentes tipos son definidos en [**7**]. El tamaño de una dirección IPv6 es de 128 bits, lo que quiere decir que se tiene un máximo de 2^{128} direcciones disponibles [**6**]. Para su

representación, los 128 bits se dividen en ocho (8) bloques de 16 bits, y cada uno de estos es convertido a cuatro dígitos hexadecimales separados por el símbolo dos puntos (“:”).

El siguiente es un ejemplo de una dirección IPv6 en formato binario:

0010000111011010000000001101001100000000000000000010111100111011
0000001010101010000000000000000000000000000000001001110001011010

La dirección es divida en 8 bloques de 16 bits cada uno:

0010000111011010 0000000011010011 0000000000000000 0010111100111011
0000001010101010 0000000000000000 0000000000000000 1001110001011010

Cada bloque de 16 bits es convertido a hexadecimal y delimitado con el símbolo dos puntos (“:”). El resultado es el siguiente:

21DA:00D3:0000:2F3B:02AA:0000:0000:9C5A

Una dirección IPv6 puede ser simplificada eliminando los ceros no significativos de un bloque, siempre dejando al menos un dígito. Finalmente es posible realizar una compresión de ceros, reemplazando bloques de ceros consecutivos con doble dos puntos (“::”). En la Figura 3.3 se puede apreciar el proceso completo:

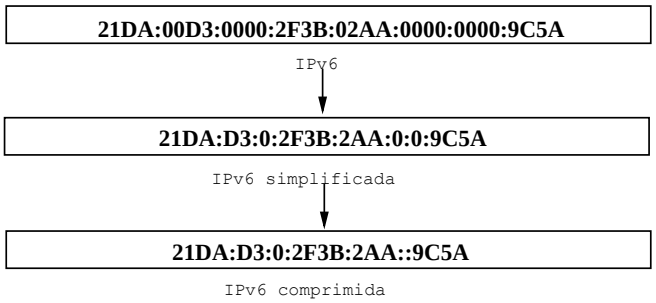


Figura 3.3: Compresión de Ceros en una Dirección IPv6

3.3.1 Prefijos en IPv6

Un prefijo es un conjunto de bits de una dirección IPv6 que se usa para identificar una subred o un tipo de dirección en específico. Estos bits tienen valor fijo y están compuestos por el conjunto de bits más significativos de la dirección IPv6. El formato de un prefijo es escrito con la notación CIDR (Classless Inter-Domain Routing) [**6**]. Ejemplo de un prefijo IPv6 de 48bits:

2001:DB8:ABBA::/48

En IPv6 el prefijo de 64bits es usado para identificar subredes y a diferencia de IPv4 este no varía en tamaño (en IPv4 el tamaño de prefijo para las subredes varia porque las direcciones IPv4 ya no son basadas en clases). Cualquier prefijo con un tamaño menor a 64bits es o una ruta resumida o un rango de direcciones IPv6 que resume el espacio de direcciones IPv6.

3.3.2 Direcciones Unicast

Es un identificador para una interfaz única. Los paquetes dirigidos a una dirección de este tipo son entregados solamente a la interfaz identificada con esta dirección.

- **Global Unicast Address:** Son direcciones globales identificadas con un formato de prefijo iniciado por los bits 001. Las direcciones globales de IPv6 son equivalentes a las direcciones públicas de IPv4, son enrutables mundialmente y alcanzables en Internet [**6**]. Su estructura está definida en [**7**] y se puede apreciar en la Figura 3.4.

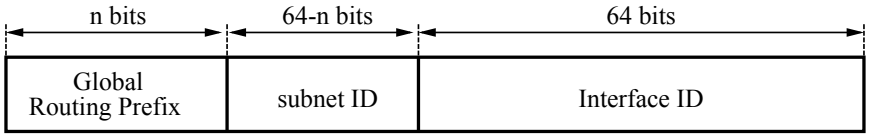


Figura 3.4: Estructura de Dirección IPv6 Global Unicast

- **Link-local Address:** Se identifican por el prefijo FE80::/64. Son usadas por los nodos al comunicarse con sus vecinos. Su alcance es la subred y nunca deben ser direccionada fuera de la misma. Pueden ser utilizadas en mecanismos de autoconfiguración (como por ejemplo por el protocolo de IPv6 "Neighbor Discovery") y en redes sin routers, por lo tanto son bastante útiles en la creación de redes temporales [**6**]. En la Figura 3.5 se muestra la estructura de una dirección de tipo Link-local.

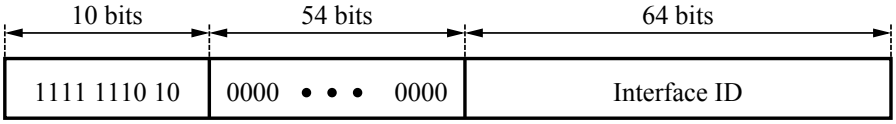


Figura 3.5: Estructura de la Dirección Link-local

Las direcciones del tipo Link-Local son auto configuradas y derivadas de la dirección MAC, aunque existe la posibilidad de configurarlas en forma manual. El proceso para obtener una dirección link-local es el siguiente (verFigura 3.6):

- Obtener el EUI-64 a partir de la dirección MAC, insertando los bytes 0xFF y 0xFE entre el tercer y cuarto byte.
- Se invierte el valor del bit "u", el cual está situado en la séptima posición más significativa. Obteniendo así la dirección EUI-64 modificada (identificador de interfaz).
- Finalmente se antepone el prefijo link-local FE80::/64 al EUI-64 obtenido previamente para obtener la dirección Link-Local.

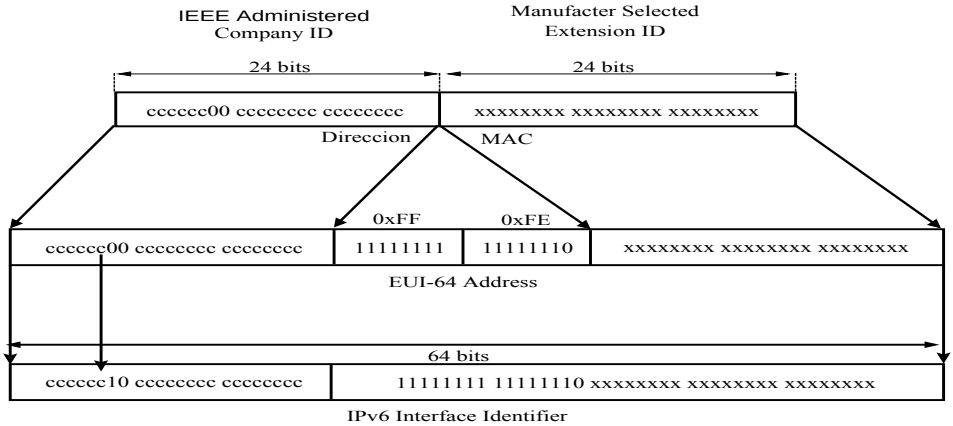


Figura 3.6: Conversión de una Dirección IEEE 802 a un Identificador IPv6

- **Site-local Address:** Las direcciones site-local son las equivalentes al espacio de direcciones privadas de IPv4. Estas direcciones no son alcanzables desde otras redes, y los routers de borde no deben retransmitir este tipo de tráfico fuera de la red de la institución. El prefijo de una dirección site-local es FEC0::/10 [**6**]. Su estructura se muestra en la Figura 3.7.

Figura 3.7: Dirección Site-local

Este tipo de direcciones se consideran obsoletas (RFC 3879 [**8**]). Fueron reemplazadas por las direcciones ULA (Unique Local Address).

- **Unique-Local Address:** Estas direcciones son únicas globalmente pero no deben ser direccionadas hacia Internet. Comúnmente llamadas "local IPv6 address" (dirección IPv6 local), son especificadas en [**7**]. Su estructura se puede apreciar en la Figura 3.8.

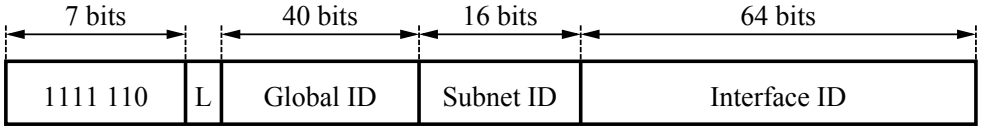


Figura 3.8: Unique-Local Address

Utilizan el prefijo FC00::/7. El bit "L" con valor 1 indica que el prefijo es asignado localmente, mientras que el bit L en 0 significa que el prefijo es asignado globalmente. Los siguientes 40 bits correspondientes al "Global ID" son usados para crear un prefijo único globalmente. Finalmente, los últimos 80 bits correspondientes a los campos "Subnet ID" e "Interface ID" identifican la subred y la interfaz, respectivamente.

- **Direcciones IPv6 especiales:** Las siguientes son direcciones IPv6 especiales:
  - Dirección no especificada: 0:0:0:0:0:0:0:0 o simplemente "::". Es usada para indicar la ausencia de una dirección (equivalente de la dirección 0.0.0.0 en IPv4). Generalmente es usada como dirección origen cuando una dirección unicast aún no ha sido determinada.
  - Dirección de loopback: 0:0:0:0:0:0:0:1 o simplemente "::1". Asignada a la interfaz de loopback, que permite auto enviarse paquetes (equivalente a 127.0.0.1 en IPv4).

- **Direcciones de transición:** IPv6 no fue diseñado para tener retro compatibilidad con IPv4, por lo tanto para ayudar en la transición de IPv4 a IPv6 y permitir la existencia de ambos tipos de hosts se definieron las siguientes direcciones y mecanismos:
  - Direcciones IPv4-compatible: 0:0:0:0:0:0:w.x.y.z o ::w.x.y.z (w.x.y.z es la representación en notación decimal punteada de una dirección IPv4 pública). Son usadas para establecer una comunicación con IPv6 a través de una infraestructura de IPv4 que utiliza direcciones públicas, tal como Internet. Este tipo de direcciones es muy raramente usado y se considera obsoleto. En la Figura 3.9 se observa su estructura.

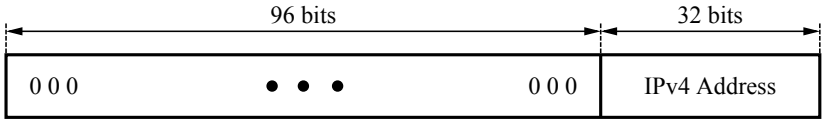


Figura 3.9: Estructura de Direcciones IPv4-Compatible

- Direcciones IPv4-mapped: 0:0:0:0:0:FFFF:w.x.y.z o ::FFFF:w.x.y.z. Son usadas para representar las direcciones de nodos IPv4 como direcciones IPv6. Un nodo IPv6 puede usar esta dirección para enviar un paquete a un nodo IPv4. En la Figura 3.10 se observa su estructura.

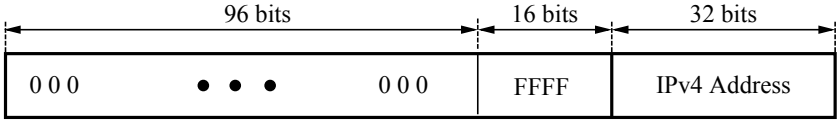


Figura 3.10: Estructura de Direcciones IPv4-Mapped

- Direcciones 6to4: 2002:WWXX:YYZZ:SubneID:InterfaceID, donde WWXX:YYZZ es la representación hexadecimal de una dirección IPv4 pública (w.x.y.z). 6to4 es una tecnología de transición que permite a hosts IPv6 comunicarse a través de una infraestructura que solo soporta IPv4. En la Figura 3.11 se observa su estructura.

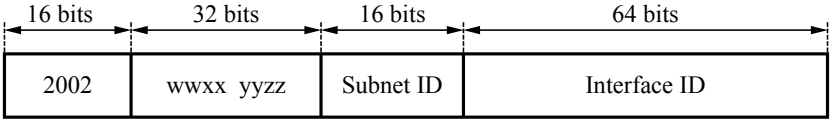


Figura 3.11: Estructura de Direcciones 6to4

- Direcciones ISATAP: 64bitsPrefix:0:5EFE:w.x.y.z, siendo w.x.y.z una dirección IPv4 privada. ISATAP es una tecnología de transición que permite a hosts doble pila (que soportan IPv4 e IPv6) comunicarse a través de una infraestructura que solo soporta IPv4. En la Figura 3.12 se observa su estructura.

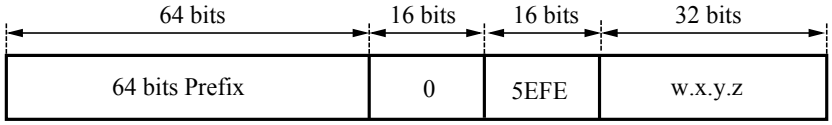


Figura 3.12: Estructura de Direcciones ISATAP

- Direcciones Teredo: Prefijo global 2001::/32. Teredo es una tecnología de transición que permite a hosts IPv6 comunicarse a través de una infraestructura que soporta solo IPv4. El resto (96 bits) se usa para codificar la dirección IPv4 de un servidor Teredo, banderas, puerto externo “oscurecido” y dirección externa “oscurecida”. En la Figura 3.13 se observa su estructura.

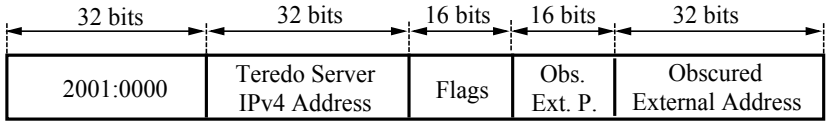


Figura 3.13: Estructura de Direcciones Teredo

3.3.3 Direcciones Multicast

Una dirección multicast es un identificador para un grupo de nodos [**6**]. En IPv6 el tráfico multicast opera del mismo modo que el tráfico multicast en IPv4. Un nodo IPv6 puede estar a la escucha de múltiples direcciones multicast. Los nodos pueden unirse o dejar un grupo multicast en cualquier momento. Las direcciones multicast no pueden ser utilizadas como direcciones origen ni como un destino intermedio, mediante el uso de la cabecera de extensión “Routing Header”.

Las direcciones multicast poseen el prefijo 0xFF, seguido de tres campos. La Figura 3.14 muestra la estructura de una dirección multicast.

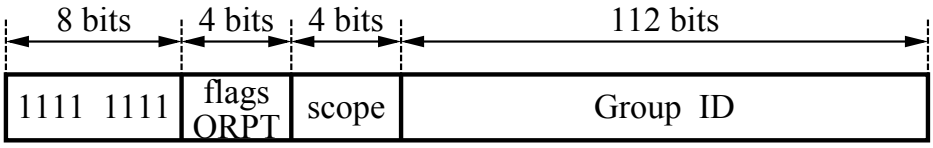


Figura 3.14: Estructura de Direcciones Multicast

El valor de los primeros 8 bits es 1. Los siguientes 4 bits son banderas utilizadas de la siguiente forma: el primer bit está reservado para uso futuro y debe ser cero (0). El segundo bit (“R”) indica si la dirección multicast embebe un “Rendezvous Point”. El tercer bit (“P”) en 1 indica que la dirección multicast posee información de prefijo. Por último el bit “T” en 1 indica si la dirección es temporal y en cero (0) indica si es una dirección asignada por la IANA.

El campo “Scope” es usado para determinar el alcance de la dirección multicast. Los posibles valores se encuentran en la Tabla 3.2.

Valor	Alcance (Scope)	Descripción
1	Interface-local	Extiende su alcance a la interfaz de un nodo.
2	Link-local	Su alcance es dentro del enlace.
4	Admin-local	El alcance más corto que debe ser administrativamente configurado.
5	Site-local	Su alcance permanece dentro del sitio.
8	Organization-local	Su alcance son múltiples sitios pertenecientes a una misma organización.
6, 7, 9 A, B, C, D	Unassigned	No han sido asignados.
E	Global	Poseen alcance global.
0, 3, F	Reserved	Están reservados, no se permite su uso.

Tabla 3.2: Valores del Campo Scope

- **Direcciones multicast conocidas:** En [**9**], se definen las direcciones multicast que han sido asignadas permanentemente. La Tabla 3.3muestra algunas de estas direcciones.

Dirección	Descripción
FF01::1	Todos los nodos en la interfaz.
FF01::2	Todos los routers en la interfaz.
FF02::1	Todos los nodos en el enlace.
FF02::2	Todos los routers en el enlace.
FF05::2	Todos los routers en el sitio.
FF02::1:2	Todos los agentes DHCPv6 en el enlace.
FF05::1:3	Todos los servidores DHCPv6 en el sitio.

Tabla 3.3: Direcciones Multicast Conocidas

3.3.4 Direcciones Anycast

Es una dirección que es asignada a múltiples interfaces. Son un intermedio entre las direcciones unicast y multicast [**6**]. Los paquetes enviados a una dirección anycast son direccionados a la interfaz más cercana que tiene asignada esa dirección anycast. Para facilitar la entrega, la infraestructura de enrutamiento debe conocer las interfaces que tienen direcciones anycast asignadas así como su distancia (en términos de métrica de enrutamiento).

Tal como se define en [**7**], las direcciones anycast solo se usan como direcciones de destino y solo son asignadas a routers. Las direcciones anycast se encuentran aún en desarrollo. Hasta ahora sólo es soportada la dirección anycast Subnet Router. Esta direcciona a todos los routers de una subred. La dirección se representa con el prefijo de subred seguido de ceros. Por ejemplo, la dirección anycast 2001:DB8:AABB:ABBA:: corresponde a los routers de la subred 2001:DB8:AABB:ABBA::/64.

3.4 Autoconfiguración de Direcciones IPv6

Uno de los aspectos más útiles de IPv6 es su capacidad de configurarse automáticamente, aun sin el uso de un protocolo de autoconfiguración (como DHCPv6 por ejemplo). Un host IPv6 puede auto configurar una dirección link-local para cada interfaz.

En IPv6 existen tres tipos de configuración:

- **Stateless o sin estado:** Configuración de direcciones y otros parámetros basada en la recepción de mensajes “Router Advertisement”. Estos mensajes poseen las banderas “Managed Address Configuration” y “Other Configuration” con valor cero (0), e incluyen una o más opciones “Prefix Information”, cada uno con la bandera “Autonomous” con valor 1 [**6**].
- **Statefull:** Configuración basada en el uso de un protocolo de configuración de direcciones (DHCPv6 por ejemplo), para obtener direcciones y otros parámetros de configuración. Un host usa autoconfiguración *statefull* cuando recibe un mensaje “Router Advertisement” con alguna de las banderas “Managed Address Configuration” y “Other Configuration” con valor igual a 1 [**6**].
- **Mixto:** Esta configuración se basa en la recepción de mensajes “Router Advertisement” que incluyen información de prefijos, cada uno con la bandera “Autonomous” con valor igual a 1, y además posee alguna de las banderas “Managed Address Configuration” y “Other Configuration” con valor igual a 1.

3.5 Ciclo de Vida de una Dirección IPv6 Auto Configurada

Completado el proceso de autoconfiguración sin estado, la dirección asignada a la interfaz puede ser utilizada hasta que expire el tiempo de vida preferido ("Preferred Lifetime") indicado en el mensaje "Router Advertisement" (RAs). En la mayoría de los casos esto no ocurre dado que nuevos mensajes RAs recibidos reiniciarán los temporizadores del tiempo de vida. Sin embargo, si no hay más mensajes RAs, el tiempo de vida expira y la dirección se convierte en obsoleta ("Deprecated"). En nuevas sesiones no se deben usar las direcciones obsoletas, sino que se debe obtener una dirección preferida si hay alguna disponible. Aun así es posible que en sesiones ya existentes continúe con el uso de alguna dirección obsoleta.

Eventualmente, el tiempo de vida válido ("Valid Lifetime") también se agotará y se removerá la dirección obsoleta de dicha interfaz. Con esto se cerrará cualquier sesión que tenga en uso esa dirección. En la Figura 3.15 se describe el ciclo de vida.

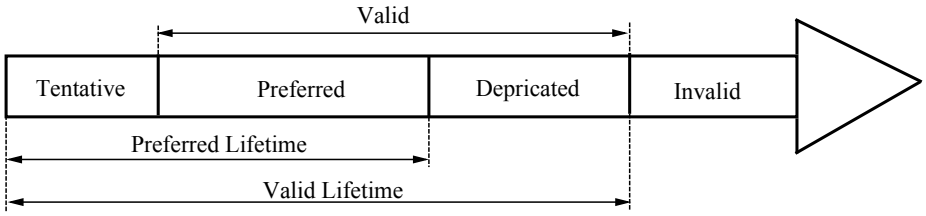


Figura 3.15: Ciclo de Vida de una Dirección IPv6 Auto Configurada

3.6 Proceso de Autoconfiguración sin Estado

La autoconfiguración sin estado (stateless address autoconfiguration) define los procesos necesarios para que un host configure las direcciones de sus interfaces basándose en información disponible localmente y la información distribuida por los routers [**6**]. Este proceso requiere una mínima configuración en los routers, no es necesario otro tipo de servidor y lo más importante es que no requiere ningún tipo de configuración manual en los hosts.

Cuando un host levanta por primera vez una interfaz ocurren los siguientes pasos:

- El host elige el identificador de una interfaz, que se deriva usualmente de la dirección MAC.
- Se crea una dirección link-local con prefijo FE80::/64 y el identificador de la interfaz.
- Se verifica que ningún otro nodo esté utilizando la dirección mediante el algoritmo Duplicate Address Detection (DAD).
- El host envía un mensaje "Router Solicitation" a todos los routers del enlace para obtener nuevos prefijos. Los routers responden con un mensaje "Router Advertisement". Estos mensajes RA contienen una lista de prefijos del enlace y un indicador en caso de que el router provea servicios de enrutamiento.
- Por cada prefijo recibido, el host configura una nueva dirección utilizando el identificador de interfaz. Verificando posteriormente su unicidad con el algoritmo DAD.
- El host se mantiene a la espera de actualizaciones sobre los prefijos del enlace y reconfigura sus direcciones en caso de que sea necesario. Los routers envían estas actualizaciones en mensajes RA no solicitados, bien sean periódicamente o cuando ocurran cambios en la configuración de la red.

La autoconfiguración sin estado no sólo provee información de los prefijos, también le indica a los hosts cual router del enlace está disponible y dispuesto a proveer servicios de enrutamiento. A diferencia de los hosts IPv4, los hosts IPv6 no tendrán un sólo router por defecto, sino una lista de routers por defecto ("default routers"). Cada vez que se trata de enviar un paquete a una dirección fuera del enlace, el host enviará el paquete a uno de los routers y dejará que ese router se encargue del paquete.

Los routers no utilizan la autoconfiguración sin estado para obtener sus direcciones; ellos necesitan ser configurados explícitamente con direcciones e información de los prefijos que deben anunciar.

3.7 DHCPv6

DHCPv6 (Dynamic Host Control Protocol Version 6) está definido en [**10**], sirve para proveer una configuración de direcciones "stateful" (direcciones adicionales) o parámetros para hosts IPv6. Un host usa un protocolo de configuración como DHCPv6, basándose en las siguientes banderas de un mensaje "Router Advertisement" enviado por un router vecino:

- **Managed Address Configuration:** También conocida como la bandera "M". Cuando es activada (posee valor 1). Esta bandera le indica al host que debe usar un protocolo de configuración con estado (DHCPv6) para obtener direcciones.
- **Other Configuration:** También conocida como la bandera "O". Cuando se encuentra activada, indica al host que debe usar el protocolo DHCPv6 para obtener otros parámetros de configuración.

La combinación de los valores de las banderas M y O son las siguientes

- **M y O ambas con valor igual a cero (0):** Esta combinación corresponde a una red sin soporte para DHCPv6. Los hosts usan la información de los RAs para autoconfiguración sin estado.
- **M y O ambas con valor igual a uno (1):** En esta combinación, DHCPv6 es utilizado para la obtención de direcciones y otros parámetros de configuración. Esta combinación es conocida como "DHCP stateful", en la cual un servidor DHCPv6 asigna direcciones a los hosts IPv6.
- **M con valor igual a cero (0) y O con valor igual a uno (1):** En esta combinación, DHCPv6 es utilizado únicamente para otros parámetros de configuración. Para las direcciones, se utiliza la autoconfiguración sin estado. Esta combinación es conocida como "DHCPv6 stateless".
- **M con valor igual a uno (1) y O con valor igual a cero (0):** En esta combinación, DHCPv6 es usado para la configuración de direcciones pero no para otros parámetros de configuración. Debido a que un host IPv6 normalmente necesita ser configurado con otros parámetros, tales como la dirección IPv6 del servidor DNS (Domain Name System), esta es una combinación poco común.

Al igual que DHCP para IPv4, los componentes de una infraestructura DHCPv6 consiste de un cliente DHCPv6 que solicita configuración, servidores DHCPv6 que proveen configuración y agentes de retransmisión DHCPv6 que retransmiten mensajes DHCPv6 entre clientes y servidores, cuando los clientes están en una subred distinta que no contiene un servidor DHCPv6.

3.8 Delegación de Prefijos

En [**11**] se describe opciones del protocolo DHCPv6, el cual provee un mecanismo para la delegación de prefijos de IPv6. Mediante estas opciones un router delegante (“delegating router”) le indica a los routers solicitantes (“requesting routers”) los prefijos que deben ser anunciados.

La delegación de prefijos con DHCPv6 es independiente de la asignación de direcciones con DHCPv6. Un router solicitante puede usar DHCPv6 sólo para delegación de prefijos o para delegación de prefijos junto con asignación de direcciones y otros parámetros de configuración.

4. **LLMNR**

LLMNR es un protocolo P2P (Peer-To-Peer), relativamente nuevo que pertenece a la familia de protocolos conocidos como “zero configuration”. Está definido en el RFC 4795 [**3**] y tal como se dijo su función consiste en resolver nombres de equipos que se encuentran en una LAN, donde no hay o no es conveniente tener un servidor DNS.

LLMNR no pretende ser un sustituto del protocolo DNS, principalmente porque opera sólo a nivel de LAN. Permite resolver nombres a direcciones IP, tanto para IPv4 como para IPv6.Reúsa el formato de paquete que emplea el protocolo DNS.

LLMNR puede coexistir con DNS y para evitar contaminar la cache DNS, las implementaciones LLMNR deben usar una cache aislada para cada interfaz LLMNR. El uso de LLMNR como mecanismo primario de resolución de nombres no es recomendado, en cambio se recomienda usar como un mecanismo secundario para dicha tarea.

Cuando es configurado como un mecanismo secundario, las consultas LLMNR sólo deberían ocurrir cuando se dan 2 condiciones:

- No existe configuración alguna de un servidor DNS.
- Todos los intentos para resolver el nombre vía DNS fueron no exitosos.

4.1 Resolución de Nombres bajo LLMNR

Las consultas LLMNR son enviadas y recibidas a través del puerto 5355 y el protocolo de capa 4 que se usa por defecto es el protocolo UDP. La dirección IPv4 en la cual escucha un host que emplea el protocolo LLMNR es la dirección multicast 224.0.0.252. Análogamente para IPv6 se tiene que la dirección de escucha es la FF02:0:0:0:0:0:1:3. La razón por la cual se utilizan estas direcciones es porque ambas son sólo de alcance local, previniendo la propagación de tráfico LLMNR entre routers. Sin embargo es posible enviar peticiones LLMNR a direcciones unicast.

Generalmente, un host es configurado tanto como Sender de peticiones LLMNR así como Responder. Es posible configurar un host que actuará sólo como Sender, sin embargo, un host configurado como Responder también debe actuar como Sender, tal y como está especificado en el RFC 4795 [**3**].

4.2 Formato de Paquetes LLMNR

LLMNR emplea el mismo formato de paquete que usa el protocolo DNS, tanto para consulta como para respuesta. El host que envía peticiones LLMNR debe encargarse de realizar la fragmentación (en caso que sea necesario).Por ende el tamaño de la petición LLMNR nunca debe exceder el tamaño del MTU del enlace. La cabecera se muestra en la Figura 4.1 (tomada de [**3**]) y se describe a continuación:

- **ID:** Representa un identificador de 16 bits, asignado por el programa que generá peticiones LLMNR. Este identificador es copiado de la consulta a la respuesta y puede ser usado por el Sender para emparejar la petición con su respectiva respuesta. El valor del ID debe ser generado de manera pseudo aleatoria para cada consulta.
- **QR (Query/Respose):** Este bit es similar al de DNS. Indica si el mensaje es de tipo consulta (QR=0), o si es de tipo respuesta (QR=1).

- **Opcode (Operation Code):** Tiene un tamaño de 4 bits e indica el tipo de consulta. Para LLMNR, el Opcode siempre está colocado en 0, tanto para consultas como respuestas.
- **C (Conflict):** El bit de conflicto. En una respuesta LLMNR si el nombre de equipo es considerado único, entonces el bit C es colocado en 0, de lo contrario, es colocado en 1. Esto sucede cuando un Responder tiene múltiples interfaces conectadas a la misma red y responde consultas encendiendo el bit para indicar posibles múltiples respuestas. Si un Responder LLMNR recibe una consulta con el bit encendido, no debe responder e inmediatamente iniciar el proceso de verificación de unicidad.

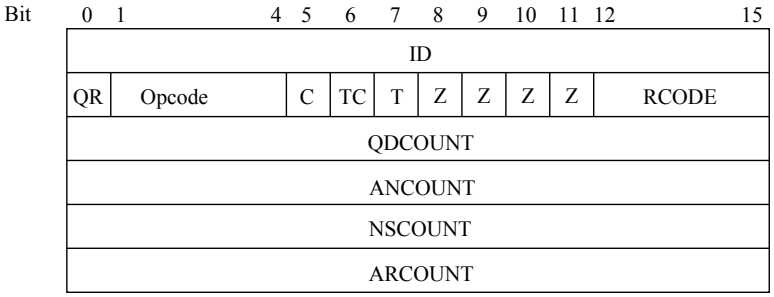


Figura 4.1: Cabecera LLMNR

- **TC (Truncation):** Cuando el bit TC está en 1, indica que la respuesta del mensaje está truncada debido a que el tamaño sobrepasa el tamaño máximo permitido en el canal de transmisión. Este bit nunca debe ser puesto en 1 en una consulta y si esto ocurriera el Responder debe ignorarlo. Si su valor está en 1 en una respuesta, el Sender deberá reenviar la consulta usando el protocolo TCP con la dirección unicast del Responder como dirección de destino.
- **T (Tentative):** El bit T es colocado en 1 en la respuesta si el Responder tiene la autoridad sobre el nombre, pero aún no ha verificado su unicidad. Un Responder debe ignorar el bit T puesto en 1en una consulta. Un Sender debe descartar silenciosamente una respuesta con el bit T fijado en 1, con la excepción de que esta sea una consulta de verificación de unicidad, en cuyo caso se considera que hay un conflicto y este debe ser resuelto por los Responders involucrados.
- **Z (Zeros):** Los bits Z están reservados para uso futuro. Toda implementación de LLMNR debe colocar estos bits en 0, tanto en consultas como en respuestas.
- **RCODE:** El código de respuesta tiene un tamaño de 4 bits. En una consulta o respuesta LLMNR (si es multicast), el Sender o Responder debe colocar RCODE en 0. Un Sender debe descartar silenciosamente una respuesta con RCODE diferente de 0.
- **QDCOUNT:** Este campo es un entero sin signo de 16 bits de longitud, el cual especifica la cantidad de entradas en la sección de pregunta. Un Sender debe colocar sólo una pregunta en dicha sección cuando realiza una consulta LLMNR. Los Responders deben descartar silenciosamente todas las consultas que tengan el campo QDCOUNT distinto de 1. Análogamente los Senders descartarán respuestas con QDCOUNT distinto de 1.
- **ANCOUNT:** Este campo es un entero sin signo de 16 bits de longitud, el cual especifica el número de registros en la sección de respuesta. Los Responders LLMNR deben descartar silenciosamente las consultas LLMNR con ANCOUNT distinto de 0.

- **NSCOUNT:** Este campo es un entero sin signo de 16 bits de longitud, el cual especifica el número de registros de tipo NS (Name Server) en la sección de registros de autoridad. Un Responder LLMNR debe descartar silenciosamente las consultas con NSCOUNT distinto de 0.
- **ARCOUNT:** Este campo es un entero sin signo de 16 bits de longitud, el cual especifica la cantidad de registros en la sección de registros adicionales.

4.3 Comportamiento del Sender

- Por defecto un Sender debería enviar consultas de nombres sencillos.
- Las consultas deben hacerse a través del protocolo UDP con puerto destino 5355 y dirección destino la dirección multicast especificada en la Sección 4.1.
- Se pueden hacer consultas para cualquier tipo de registro válido DNS.
- Si se reciben múltiples respuestas válidas con la bandera C colocada en 1 estas deben concatenarse y tratarse como una única respuesta válida.
- Si se recibe una respuesta con la bandera TC colocada en 1, entonces se debe repetir la consulta haciendo la pregunta a la dirección unicast de la respuesta y esta debe hacerse a través del protocolo TCP.
- Si se recibe una respuesta con la bandera T igual a 1 entonces se debe descartar silenciosamente.
- Se debe considerar como posibilidad no recibir una respuesta, en cuyo caso se trata como si el nombre no existiera.

4.4 Comportamiento del Responder

Un Responder generalmente deriva sus correspondientes registros (A, AAAA, PTR) para poder responder las consultas LLMNR referentes a estos registros.

Todos los hosts LLMNR son autoritativos sólo de su(s) nombre(s) y su(s) FQDN (Fully Qualified Domain Name). Por ejemplo el host LLMNR con nombre “host1” y perteneciente al dominio “example.com” es autoritativo sólo del nombre “host1” y “host1.example.com”.

Es responsabilidad del Responder asegurar que los registros a retornar en las respuestas LLMNR deben incluir sólo valores que son válidos en la correspondiente interfaz local, como por ejemplo direcciones IPv4 e IPv6 válidas en el enlace local, así como nombres autoritativos que han pasado el proceso de verificación de unicidad.

- Se debe escuchar en el puerto UDP 5355 en la dirección de multicast definida en la Sección 4.1, y en el puerto TCP 5355 para una dirección unicast que haya sido asignada a la interfaz, la cual podrá ser colocada como dirección origen en las respuestas LLMNR.
- Se debe responder directamente al puerto desde el cual fue enviada la consulta tanto para el protocolo TCP como el protocolo UDP.
- Las respuestas siempre deben ser enviadas desde el puerto al cual fueron dirigidas.
- La respuesta debe ser enviada como un unicast.
- Se debe responder sólo consultas de nombres y direcciones en las cuales el Responder tiene la autoridad.
- Si un Responder detecta que se encuentra conectado a la misma red a través de múltiples interfaces debe responder con el bit C colocado en 1.
- No se debe responder usando datos extraídos desde la cache DNS.
- Si un Responder tiene la autoridad sobre un nombre, debe responder con el campo RCODE igual a 0 (RCODE=0) y la sección de respuesta debe estar vacía

en aquellas consultas en las cuales se solicite un registro que el Responder no posee.
- Las consultas unicast hechas a través del protocolo UDP deben ser descartadas silenciosamente.
- Antes de realizar exitosamente el proceso de verificación de unicidad un Responder debe enviar una respuesta con el bit T colocado en 1.

Cuando múltiples direcciones representan una respuesta válida a una consulta LLMNR, el Responder debe seleccionar una basado en el siguiente algoritmo:
- Si la dirección origen de la consulta LLMNR es una dirección de tipo Link-Local, entonces se debe seleccionar para la respuesta una dirección del mismo tipo (Link-Local), si esta se encuentra disponible.
- Si la dirección origen de la consulta LLMNR es una dirección del tipo global enrutable, entonces debe seleccionarse para la respuesta una dirección del mismo tipo (global enrutable), si esta se encuentra disponible.

4.5 Retransmisión y Jitter

Para la retransmisión existe un timer llamado “LLMNR_TIMEOUT” que es usado para determinar si un Sender debe o no retransmitir una consulta. Si una consulta hecha con UDP no es respondida y expira el intervalo del temporizador, entonces debe ser repetida, hasta un máximo de 3 veces.

Cuando la consulta es hecha con el protocolo TCP la retransmisión es manejada por este protocolo, por lo tanto el temporizador queda sin efecto. Cuando la consulta es realizada con el bit C puesto en 1, debe ser hecha con UDP y no debe retransmitirse nunca.

Un Sender considera que una consulta está completa cuando recibe la primera respuesta dentro del intervalo de LLMNR_TIMEOUT con el bit C puesto en 0. En caso que tenga el bit C colocado en 1 entonces el Sender debe esperar el timer LLMNR_TIMEOUT + JITTER_INTERVAL, con el fin de recoger todas las posibles respuestas.

Para evitar la sincronización sobre la red, tanto las consultas como las respuestas LLMNR deben hacerse en un intervalo de tiempo entre 0 y JITTER_INTERVAL. El valor sugerido para LLMNR_TIMEOUT es 100 ms para Ethernet (incluido 802.11) o 1 segundo si LLMNR es configurado manualmente. Para JITTER_INTERVAL se sugiere 100ms.

4.6 Verificación de Unicidad de Nombre

Antes de enviar respuestas con el bit T colocado en 0, un Responder configurado con un nombre único debe validar que no exista otro host en la misma red que responda al mismo nombre, esto se conoce como el proceso de verificación de unicidad. Una vez realizado el proceso con éxito, el Responder puede considerarse autoritativo del nombre verificado y puede responder a consultas con el bit T colocado en 0.

El proceso de verificación de unicidad se realiza sólo en los siguientes casos:
- En el inicio o reinicio.
- En el regreso de un estado de inactividad.
- Cuando es configurado para responder las consultas LLMNR en una interfaz que permita el envío y recepción de tráfico IP.
- Cuando se adquiere una nueva IP y configuración en una interfaz.
- Cuando se detecta un conflicto de nombre.

Para realizar el proceso, el Responder debe enviar una consulta del nombre a verificar con el bit C puesto en 0, a través de todas las interfaces por las cuales responde consultas LLMNR, es recomendable realizar la consulta para cualquier tipo de registro (type="ANY").

Si ninguna respuesta es recibida entonces debe retransmitir la consulta tal como se especifica en la Sección 4.5(Retransmisión y Jitter). Si se recibe una respuesta entonces el host debe validar que la dirección origen en la respuesta no coincida con la dirección de cualquiera de sus interfaces, en cuyo caso no habría conflicto. Si la respuesta recibida no proviene de alguna de las interfaces del host, entonces debe verificar si el bit T en la respuesta es igual a 0, en cuyo caso este host no debe responder a consultas del nombre a verificar.

Si la respuesta recibida tiene el bit T en 1 entonces el host debe verificar si la dirección IP origen recibida en la respuesta es lexicográficamente menor a la dirección IP origen en la consulta, en cuyo caso este host no debe responder a consultas del nombre a verificar.

4.7 Detección y Resolución de Conflictos

Para permitir la detección y posterior resolución de conflictos, un host que recibe múltiples respuestas (con el bit C puesto en 0) a una consulta LLMNR debe advertir a los demás hosts en la red que existe un conflicto, enviando una consulta por el nombre en conflicto con la bandera C colocada en 1.

Las consultas con el bit C puesto en 1 son consideradas advertencias y los Responders involucrados deben verificar inmediatamente la existencia del conflicto. La consulta con el bit C en 1 no debe ser respondida. Para poder detectar si existe conflicto con su nombre, los Responders involucrados deben generar una consulta con su mismo nombre, tipo y clase junto con la bandera C puesta en 0.

Si una respuesta es recibida, entonces se debe verificar si la dirección origen de la respuesta coincide con la dirección origen de la consulta, en cuyo caso se considera que no hay conflicto ya que la respuesta viene del mismo host que originó la pregunta. En caso contrario (la respuesta tiene la bandera C igual a 0) se ha detectado un conflicto, a lo cual se debe dejar de responder inmediatamente a consultas para el nombre en conflicto y realizar el proceso de verificación de unicidad de nombre (descrito en la Sección4.6) para resolver el conflicto.

4.8 Configuración de LLMNR

LLMNR puede ser configuración de forma manual o automática en cada una de las interfaces. Por defecto LLMNR debe estar habilitado en todas las interfaces en todo momento y en caso de que no esté configurado, el host no debe escuchar ni enviar consultas.

Es posible configurar LLMNR a través de DHCP (Dynamic Host Configuration Protocol) haciendo uso de la opción "LLMNR_Enable" la cual permite explícitamente activar o desactivar LLMNR en una determinada interfaz.

4.9 Configuración en Múltiples Interfaces

Existe la posibilidad de configurar LLMNR en múltiples interfaces, cuando este sea el caso se debe mantener una caché independiente por cada interfaz de red que use LLMNR.
En el caso que un host mantenga multiples interfaces que usen LLMNR.Su nombre de equipo designado será el mismo para todas las interfaces del host y responderá consultas en todas las interfaces que tengan LLMNR activado.

Este host chequeará la unicidad de su nombre (y lo defenderá de ser necesario) en todas las interfaces que tengan LLMNR activado.

Tal como se muestra en la Figura 4.2, existen 2 hosts en la red “A” con el nombre “PC1” (izquierdo y derecho), es posible que ambos se adueñen del nombre si alguno de los 2 es desconectado de la red “A”.Cuando se detecte el conflicto el PC1 derecho no podráresponder consultas en esa red. Sin embargo sí podrá responder consultas en la red “B”.

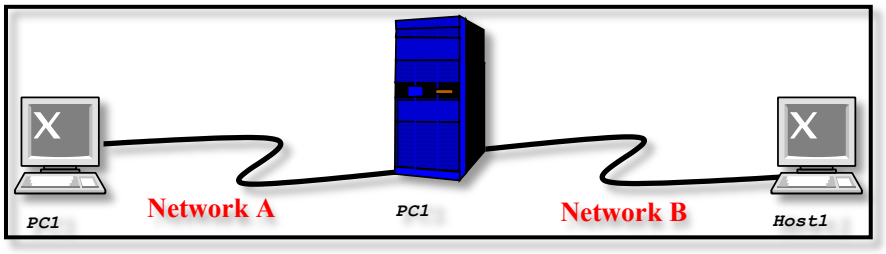


Figura 4.2: Host con Múltiples Interfaces

Dado que los nombres de equipo son únicos en cada red, podría darse el caso en que distintos hosts en distintas redes usan el mismo nombre de equipo, esta situación se muestra en la Figura 4.3.

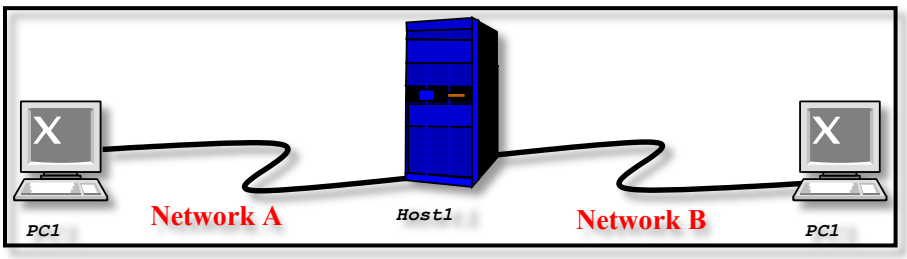


Figura 4.3: Conflicto de Nombre en Redes Distintas

Si el host “Host1” lanza una consulta del nombre “PC1” lo hará por ambas interfaces (red A y B), recibirá respuesta por las 2 interfaces. En este caso el “Host1” no sabrá distinguir cual es la respuesta deseada y su comportamiento será definido por la implementación. Este escenario muestra que el mecanismo de resolución de conflictos no funciona entre hosts que se encuentran en redes distintas.

4.10 Seguridad en LLMNR

Debido a que LLMNR es un protocolo diseñado para ser usado localmente limita ataques de Senders que no se encuentren en la red, sin embargo puede existir la posibilidad de que un Responder externo logre llegar a un Sender.

Uno de los ataques más comunes es la Negación de Servicio (DoS). Este se logra usando el mismo nombre de host en diferentes equipos, de esta manera se crea un conflicto en la red y no se pueden enviar o recibir peticiones. También se puede lograr hacer pasar por un host determinado por otro, y recibir el tráfico que iba dirigido al host original.

LLMNR está diseñado para impedir consultas externas a la red, sin embargo es posible que algunos routers hagan una mala implementación del multicast de alcance local, permitiendo una fuga de paquetes multicast. Esta situación podría representar un problema de seguridad y para prevenir posibles conexiones externas (TCP SYN, TCP SYN ACK) todas las consultas y respuestas LLMNR deben ser generadas con el campo TTL (IPv4) y Hop Limit (IPv6) colocado en 1.

Un hipotético atacante que se encuentre dentro de la red podría hacer “Spoofing” si logra responder más rápido que un Responder autoritativo, logrando que la respuesta legítima no sea considerada por el host que generó la consulta, debido a que LLMNR considera una consulta respondida una vez que recibe la primera respuesta válida.

Existen varias maneras de aumentar la seguridad en LLMNR, entre ellas se tienen:

- Usar un algoritmo para generar números aleatorios para el campo ID.
- Limitar el uso de LLMNR a resolver únicamente nombres sencillos.
- Implementar autenticación

5. **Trabajos Relacionados**

Existen protocolos similares y previos a LLMNR que se encargan (entre otras cosas) de resolver los nombres de equipos dentro de una red local. En este capítulo se abordan varios de estos protocolos.

5.1 NetBIOS (Network Basic Input Output System)

Es un protocolo creado por IBM, el cual define un API (Application Programming Interface) para la comunicación de aplicaciones a través de la red. NetBIOS provee servicios de sesión y transporte del modelo OSI (Open Systems Interconnection), además tiene 2 modos de comunicación: sesión o datagrama [**12**].

El modo de sesión funciona de manera similar al protocolo TCP, es decir que provee funciones como intercambio de datos de mayor tamaño y control de errores y retransmisión. Las sesiones se establecen mediante un intercambio de paquetes, el equipo que desee establecer la conexión envía la solicitud y esta debe ser respondida, luego el equipo envíala solicitud para establecer la sesión. Análogamente el modo de datagrama funciona similar al protocolo UDP permitiendo una comunicación sin conexión.

NetBIOS puede operar sobre el protocolo TCP/IP y provee 3 servicios básicos: (1) Name, (2) Session y (3) Datagram. De estos 3, el servicio “Name” sirve para hacer resolución de nombres NetBIOS [**12**]. Trabaja con en el puerto 137 y al igual que LLMNR hace reutilización de la estructura de paquete DNS para la resolución de dichos nombres.

La asignación de nombres NetBIOS tiene pocas restricciones, por ejemplo caracteres no alfanuméricos pueden ser usados para los nombres. Sin embargo algunas implementaciones de NetBIOS podrían no soportar esto. Existen 3 maneras de realizar el mapeo de nombres NetBIOS a una dirección IP dentro de una red:

- Cuando no se consigue la dirección de un equipo en memoria caché, se emite un mensaje broadcast y el equipo que tiene el nombre responde con su dirección.
- Dentro del archivo `Lmhosts` se encuentran definidos los nombres de equipos con sus direcciones asociadas.
- Haciendo uso de un servidor NBNS (NetBIOS Name Server), el cual está encargado de realizar el mapeo de las direcciones con los nombres.

Para que un equipo se pueda comunicar con otros dispositivos, este debe ser identificado previamente de manera única. Los nombres en NetBIOS tienen un tamaño de 16 bytes y a veces las aplicaciones usan el Byte 16 para indicar algún servicio en particular [**12**]. En la Tabla 5.1se pueden apreciar algunos de los servicios NetBIOS.

16th Byte (En Hexadecimal)	Descripción
00	Estación de trabajo.
01	Servicio de mensajería.
20	Servicio de compartición de archivos.
2B	Servidor de Lotus Notes.

Tabla 5.1: Ejemplos de Sufijos NetBIOS

Los nombres en NetBIOS no son jerárquicos y no poseen subdivisión. Debido a que no existe una manera de reconocer las redes que soportan NetBIOS, los protocolos que trabajan con NetBIOS no pueden ser enrutables.

El acceso a la interfaz de conexión de NetBIOS se hace a través de la interrupción 5Ch.Estáinterrupción es llamada con los registros ES:BX apuntando a una estructura de 64bytes, que es conocida como el NCB (Network Control Block), la cual contiene los datos requeridos como nombres, comandos, apuntadores, etc. El NCB no puede ser modificado hasta que un comando sea completado, así que no puede ser usado por otros comandos a la vez, sin embargo una vez que se termine de ejecutar el comando se puede modificar.

La estructura del NCB contiene los siguientes campos:

- **Byte 0 –** Command: Se trata de un código de comando el cual hace referencia al código que se desea ejecutar.
- **Byte 1** – Return Code: Devuelve el código de competición para un comando síncrono, o el código de iniciación para un comando asíncrono.
- **Byte 2** – Local Session: Especifica el número de sesión para un comando. El número de sesión es retornado cada vez que una sesión se abre.
- **Byte 3** – Name Number: Específica el número único para el nombre utilizado, cada vez que se añade un nombre, se devuelve un número único.
- **Bytes 4-7** – Buffer Address: Especifica la dirección para el buffer que se usa para transmitir y recibir comandos.
- **Bytes 8-9** – Buffer Length: Especifica el tamaño del buffer.
- **Bytes 10-25** – Call Name: Nombre del host local o remoto que se desea contactar.
- **Bytes 26-41** – Local Name: Nombre del host local, es requerido solamente cuando se agrega o se elimina el nombre.
- **Byte 42** – Receive Timeout: Tiempo de espera para recibir comandos.
- **Byte 43** – Send Timeout: Tiempo de espera para enviar comandos.
- **Byte 44-47** – POST Address: Especifica la dirección de la rutina POST para comandos asíncronos.
- **Byte 48** – Network Number: Si existe más de un adaptador en el host, se debe especificar el número que se desea usar.
- **Byte 49** – Command Complete: Este es el código de la competición de los comandos asíncronos. No tiene ninguna función para los comandos síncronos, ya que el control sólo se devuelve una vez que se han completado, y el código de terminación se devuelve en el campo return_code (1 byte).
- **Bytes 50-63** – Reserved: Estos bytes deben estar presentes, y no deben ser cambiados. El software de red utiliza estos para espacio de trabajo.

Existen 2 tipos de comandos en NetBIOS, los síncronos se inician con la creación del NCB y llamando a `5Ch-interruption` con `ES:BX` apuntando al NCB. El control será retornado al programa cuando el comando se haya completado en su totalidad o haya expirado el tiempo de espera. El bit 7 del comando se establece en 0 para indicar que es un comando síncrono. Cuando el control es retornado, el campo return_code indica la iniciación o el estado de terminación.

El comando asíncrono se inicia de la misma manera que el síncrono pero estableciendo el bit 7 en 1. El control es retornado inmediatamente y se debe verificar cuando el comando sea completado, para esto existen 2 maneras:

- Sondear el campo command_complete del NCB: si contiene `FFh` el comando aún se está ejecutando.

- Una rutina POST: esta es una rutina de usuario que es llamada por un software con el `ES:BX` apuntando al campo command_complete.

Los comandos asíncronos pueden ser usados para operaciones de multitarea. Un comando asíncrono puede ser iniciado dentro de la rutina después que esta haya iniciado otro comando, por ejemplo, una vez que un mensaje ha sido recibido en una charla, el NCB debe ser reinicializado para recibir el siguiente mensaje. Sin embargo, los comandos síncronos no pueden, ni deben, ser llamado en una rutina POST.

5.2 WINS (Windows Internet Name Service)

Es un servicio encargado del registro y resolución de nombres, el cual conecta equipos usando NetBIOS con direcciones IP de Internet. Cuando hay servidores WINS dentro de una red, los usuarios pueden acceder a recursos de la red por su nombre. Además, el software y los servicios que se ejecutan en los computadores y otros dispositivos pueden realizar consultas de nombres en su servidor WINS para resolver nombres a direcciones IP. Un servidor WINS provee ciertos beneficios como lo son:

- Proveer una base de datos dinámica donde se tiene la dirección IP asociada a un nombre.
- Tener una sola base de datos centralizada que permite no depender del archivo `Lmhosts` para la resolución de nombres.
- Reducir los mensajes de broadcast que usa NetBIOS, ya que se consultaría directamente en la base de datos.

Los componentes básicos de WINS son: un servidor y un cliente. A veces es necesario usar un proxy WINS. Un servidor WINS está encargado de manejar los registros de nombres provenientes de los clientes y responder a las solicitudes de NetBIOS para retornar las direcciones IP de existir en la base de datos. Cuando se unen a la red, los clientes WINS intentan registrar su nombre. Una vez registrados, los clientes pueden realizar las consultas al servidor para la resolución de dirección.

Cuando un equipo está en una red en la cual no puede tener acceso al servidor WINS, se usa un proxy. Un proxy WINS es el encargado de hacer un puente entre el equipo y el servidor WINS para que este pueda revolver el nombre.

5.3 Multicast DNS (mDNS)

Es un protocolo que provee un servicio de sistemas de nombres fácil de configurar y mantener para equipos dentro de una LAN. Todos los equipos dentro de la red proveen funciones estándares de DNS, haciendo uso de multicast en vez de usar un servidor DNS unicast. Este protocolo fue propuesto y desarrollado por Apple Inc. [**13**].

Las consultas de mDNS son enviadas a la dirección IPv4 224.0.0.251, la cual está reservada para mDNS. Análogamente la dirección IPv6 de mDNS es FF02::FB. El puerto destino que usa es el 5353. Debido a que tanto la dirección IPv4 e IPv6 son de alcance local, las consultas de mDNS no salen de la red local. Al igual que LLMNR, el algoritmo que emplea mDNS para la apropiación de nombres y resolución de conflictos reemplaza la figura de una autoridad centralizada por la de nodos cooperativos. También hace una reutilización del formato de paquete del protocolo DNS.

5.3.1 Nombres mDNS

Cualquier consulta DNS que termine con `.local` debe ser enviada a algunas de las direcciones multicast (según sea el caso) definida en la Sección5.3. Las consultas DNS que no terminen con el sufijo ".local" pueden ser enviadas a la dirección multicast de mDNS, solo en caso de que no exista un servidor DNS disponible.

No hay restricción de ningún tipo a la hora de seleccionar un nombre o etiqueta de host, excepto en cuanto al tamaño, el cual posee un tamaño máximo de 63 bytes. El nombre autoritativo de un host mDNS tendrá la siguiente forma: "nombre-seleccionado-o-etiqueta local". El dominio ".local" es especial porque solo hace referencia a los nodos dentro de la red local.

5.3.2 Comprobación de Unicidad de Nombre

Una vez que un host seleccione (o se le asigne) un nombre, este deberá hacer una comprobación de unicidad del mismo, a través de una consulta mDNS para el nombre seleccionado. Si ninguna respuesta es recibida entonces se repite la consulta 250 ms después de haber enviado la primera. Si no se recibe respuesta para la segunda consulta entonces se intenta una tercera 250 ms después de enviada la segunda y se espera una respuesta por 250 ms. Después de haber enviado 3 consultas y esperar un total de 750 ms, si ninguna respuesta es recibida entonces el host ha verificado exitosamente el nombre y es autoritativo del mismo.

Si en algún punto del proceso de verificación se obtiene una respuesta, entonces el host que está haciendo la comprobación de unicidad deberá seleccionar un nuevo nombre.

5.3.3 Consultas en mDNS

Existen 2 tipos de consulta en mDNS, la consulta de tipo "One-Shot" emitidas siguiendo el esquema de resolución DNS, y las consultas "Multicast DNS continuas" basadas en consultas DNS Multicast que soportan operaciones asíncronas.

Las consultas One-Shot de mDNS se realizan enviando peticiones a la dirección 224.0.0.251 en el puerto 5353, o la dirección IPv6 FF02::FB. Este tipo de consulta se usa para resolver consultas básicas, y se considera como respuesta válida a la consulta, la primera que llegue. Generalmente este tipo de consultas se hace usando un número aleatorio como número de puerto UDP origen (aunque también se podría usar uno no aleatorio), sin embargo es importante que la consulta nunca use como puerto UDP origen el número 5353, ya que significaría que se está enviando una consulta completa de mDNS (y no una de tipo One-Shot)

A diferencia de las consultas One-Shot, las consultas del tipo continuas no terminan al recibir una respuesta (son más asíncronas), es decir, dependiendo de la consulta, al recibir una respuesta se puede seguir realizando consultas de manera consecutiva, hasta que el host emisor considere que no hacen falta más respuestas. Determinar cuándo no se necesitan más respuestas varía según la operación que se realiza.

El intervalo de tiempo entre la primera y segunda consulta debe ser de al menos 1 segundo. Para las siguientes consultas, el tiempo entre estas debe aumentar sucesivamente en por lo menos un factor de 2, y el host (que emite las consultas)debe aplicar el mecanismo de "supresión de respuesta conocida" (The Known-Answer

Suppression). Este último informa a los hosts Responders cuáles respuestas son ya conocidas, evitando así desperdicia rancho de banda al retransmitir respuestas repetidas. Las consultas de este tipo deben ser enviadas y recibidas a través del puerto 5353. Se puede tener un caso en el cual un host emite varias consultas en el mismo mensaje. La semántica de un mensaje con múltiples consultas es equivalente a las consultas continuas de mDNS, donde por cada mensaje se emite una consulta. El uso de esta técnica es para optimizar la eficiencia de las consultas.

5.3.4 Respuestas en mDNS

Cuando un Responder de mDNS envía un mensaje, la sección de recursos del mensaje debe contener sólo los registros para los cuales este es autoritativo. Un Responder mDNS no debe responder usando la data guardada en memoria caché, la cual ha sido obtenida de otros Responders en la red.

La manera como se determina si un registro corresponde a una consulta es usando las reglas DNS, el nombre del registro debe coincidir con el nombre en la consulta, el registro `rrtype` debe coincidir con la consulta en el campo `qtype`, y el campo `rrclass` debe coincidir con el campo `qclass` de la consulta.

5.3.5 Resolución de Conflictos

Un conflicto ocurre cuando un Responder mDNS contiene un registro único el cual es autoritativo y recibe un mensaje mDNS que contiene el mismo nombre, `rrtype` y `rrclass` pero en el campo `rdata` contiene datos inconsistentes.

Cuando un Responder mDNS recibe una respuesta y detecta que hay conflicto, el Responder debe inmediatamente restablecer el registro en conflicto a “estado de sondeo”, en este estado se determina quién es el ganador y perdedor del nombre.

Al volver al estado de sondeo, el host que desea un nombre de host exclusivo realizará los pasos necesarios para asegurar la obtención de un nombre de host único. Algunos de estos pasos son cambiar el nombre del equipo para encontrar un nombre único dentro de la red, una vez encontrado y verificado establecer el nombre en el equipo deseado.

5.3.6 Verificación de la Dirección de Origen

Todas las respuestas de mDNS (incluyendo las unicast), deben ser enviadas con el campo TTL (de la cabecera IP) en 255para tener compatibilidad con otras versiones de mDNS. Un host sólo debe aceptar respuestas a consultas que se generaron en la red local, y silenciosamente debe descartar cualquier otra que le llegue. Existen 2 maneras para conocer si la respuesta fue generada dentro de la red local:

- Si la respuesta recibida tiene como dirección destino 224.0.0.251 o FF02::FB, entonces la consulta tuvo que ser originada localmente, sin importar la dirección IP origen.
- Para las direcciones IPv4, si la respuesta recibida tiene una dirección IP de destino del tipo unicast, entonces se hace una comparación bit-a-bit entre la dirección IP de la interfaz de red junto con la máscara de subred y esta se compara bit-a-bit con la dirección IP y máscara de subred de la respuesta. Para las direcciones IPv6 se usa cualquiera de los prefijos en la dirección IPv6 en la interfaz donde se recibe el paquete, los primeros n bits del prefijo de la dirección de origen coinciden con los primeros n bits de la dirección local.

Puesto que los hosts que generan consultas van a ignorar respuestas originadas fuera de la red, un Responder debe evitar generar respuestas que probablemente serán descartadas.

5.3.7 Formato de Paquete mDNS

Los paquetes mDNS que se envían usando el protocolo UDP pueden tener un tamaño máximo igual al valor del MTU (Maximum Transfer Unit) de la interfaz, menos el espacio necesario para la cabecera IP y la cabecera UDP. En caso de que un mensaje mDNS sea muy grande, el Responder debe enviar la respuesta usando múltiples fragmentos.

Un paquete mayor que el MTU establecido (el cual debe ser enviado de manera fragmentada) no debe contener más de un registro e incluso cuando es fragmentado no debe tener un tamaño mayor a 9000bytes.

Los campos del mensaje mDNS son los siguientes:

- **ID(Query Identifier):** mDNS debe poder escuchar mensajes no solicitados emitidos por hosts que se están encendiendo. Debido a que estos mensajes pueden tener respuestas a consultas que están esperando ser respondidas, mDNS debe examinar todas las respuestas sin importar el contenido del campo ID (Para mDNS el campo ID no es tan importante conocerlo).
- **QR(Query/Response) Bit:** En una consulta el bit QR debe ser 0 y en la respuesta este bit debe ser colocado en 1.
- **OPCODE:** Tanto en consultas como respuestas debe ser 0, si un mensaje contiene un valor diferente debe ser descartado silenciosamente.
- **AA(Authoritative Answer) Bit:** En consultas el bit debe ser 0, y debe ser ignorado cuando se recibe con el valor puesto en 1.
- **TC(Truncated) Bit:** Si el bit está puesto en 1 en la consulta, significa que más respuestas vendrán a continuación. Un Responder debe registrar este hecho y esperar por las respuestas adicionales. En las respuestas, el bit debe ser 0 y debe ser ignorado cuando se recibe con el valor puesto en 1.
- **RD(Recursion Desired) Bit:** Debe ser 0 en ambos casos (consulta y respuesta) y debe ser ignorado cuando se recibe con el valor puesto en 1.
- **RA(Recursion Available) Bit:** Debe ser 0 en ambos casos (consulta y respuesta) y debe ser ignorado cuando se recibe con el valor puesto en 1.
- **Z(Zero):** Debe ser 0 en ambos casos (consulta y respuesta) y debe ser ignorado cuando se recibe con el valor puesto en 1.
- **AD(Authentic Data) Bit:** Debe ser 0 en ambos casos (consulta y respuesta) y debe ser ignorado cuando se recibe con el valor puesto en 1.
- **CD (Checking Disabled) Bit:** Debe ser 0 en ambos casos (consulta y respuesta) y debe ser ignorado cuando se recibe con el valor puesto en 1.
- **RCODE(Response Code):** Debe ser 0 en ambos casos (consulta y respuesta) y debe ser descartado silenciosamente en caso de tener un valor diferente a 0.

5.4 Microsoft LLMNR (ms-llmnr)

El modelo de implementación LLMNR de Microsoft se basa de manera parcial en el RFC4795, no implementa el protocolo en su totalidad. La especificación de su modelo está descrita en [**4**].

5.4.1 Modelo de Datos del Sender

El estado que debe mantener cada Sender es similar al descrito en elRFC4795 [**3**], sin embargo el RFC declara en su Sección 5.4, que "las implementaciones LLMNR deben usar una caché distinta y aislada para cada interfaz LLMNR". Esta especificación es vaga en cuanto a si LLMNR debe (obligatoriamente) soportar caché o si LLMNR debe tener una caché (de existir) distinta y aislada de la caché de DNS. En base a esto, tal como se especifica en [**4**], las implementaciones basadas en el modelo de Microsoft pueden soportar caché. También hay soporte para caché negativa (negative caching) y esta sólo debe suceder cuando ya exista una entrada por una consulta errónea (para el nombre solicitado) en la cache DNS.

5.4.2 Temporizadores

La Sección 2.7 del RFC4795 [**3**] especifica que para evitar errores de sincronización, la trasmisión de cada consulta LLMNR se debe retrasar por un tiempo aleatorio en un intervalo que va desde 0 hasta "Jitter_Interval" (100 ms). Para implementar este comportamiento es requerido hacer uso de un temporizador. En el modelo de Microsoft, el Sender envía las consultas inmediatamente sin ningún tipo de retraso, descartando el uso de un temporizador.

5.4.3 Inicialización

ms-llmnr no especifica el uso y el soporte a NSSO (Name Service Search Option) y a la opción "LLMNR_Enable", sin embargo en el RFC4795 [**3**], se incluyen estos solo como referencia, indicando que no son necesarios para la implementación de LLMNR. En *ms-llmnr* no hay requisitos de conformidad con respecto a esas referencias.

5.4.4 Mensaje de Procesamiento de Eventos y Reglas de Secuencia

La Sección 2.1 del RFC4795 [**3**] especifica que un Sender debe aceptar respuestas tan grandes de tamaño como el menor entre el tamaño del MTU del enlace y 9194 octetos. En *ms-llmnr* un Sender debe aceptar respuestas tan grandes como el tamaño máximo de la carga en UDP que pueda ser transmitida tanto en IPv4 como en IPv6. El Sender debe especificar el valor de la carga que soporta UDP.

Cuando una respuesta llega con el bit TC en 1, se recomienda que LLMNR descarte la respuesta y reenvíela consulta usando TCP. En la implementación de Microsoft [**4**], el Sender tiene la opción de enviarlo de esa manera, sin embargo debe ignorar el bit TC y procesar la respuesta como si no existiera truncamiento.

El RFC4795 [**3**] especifica que puede ocurrir que se reciba una consulta sin el bit C seguido por un mensaje con el bit C, un Sender debe estar preparado para manejar los conflictos incluso después de haber respondido la consulta. En *ms-llmnr* el Sender debe procesar las respuestas adicionales una vez que se haya respondido la consulta.

La Sección 2.4 del RFC4795 [**3**] recomienda que un Sender LLMNR envíe consultas PTR usando TCP unicast diferentes de UDP que usa multicast. En *ms-llmnr* un Sender puede usar unicast para TCP para las consultas PTR, sin embargo es conveniente que use UDP multicast.

El RFC4795 [**3**] no especifica si las consultas de nombre deben ser enviadas usando "UTF-8" (RFC3629) o "Punycode" (RFC3492). En *ms-llmnr*, el Sender envía las consultas usando UTF-8.

5.4.5 Responder LLMNR

La Sección 2.1 del RFC4795 [**3**] recomienda que un Responder solo envíe por UDP respuestas usando el máximo tamaño permitido por UDP para evitar fragmentación. En *ms-llmnr* un Responder debe enviar respuestas usando UDP con el máximo tamaño que permita la carga en UDP, si la respuesta no entra en un solo mensaje UDP entonces el Responder debe poner la mayor cantidad de registros en el mensaje y enviar la respuesta sin marcar el bit TC.

El Responder debe responder consultas de tipo A, AAAA,PTR y ANY. El Responder puede responder otro tipo de consultas, sin embargo, debe descartarlas silenciosamente para cualquier otro tipo de consulta. Para los tipos de consulta ANY el Responder debe retornar registros de tipo A o AAAA.

La Sección 4.2 del RFC4795 [**3**] especifica que un Responder debe guardar en un log los conflictos de nombre detectados. En *ms-llmnr* [**4**] puede que el Responder los guarde o no.

6. **Marco Metodológico**

Para alcanzar los objetivos planteados en el Capítulo 2 hace falta definir un esquema de trabajo o metodología de trabajo que permita el desarrollo rápido y eficiente de los requerimientos de la aplicación. A continuación se presenta la especificación de la metodología empleada así como otros detalles importantes tomados en cuenta para el desarrollo del demonio.

6.1 Adaptación de la Metodología de Desarrollo

La mayoría de las metodologías de desarrollo que existen plantean un esquema de trabajo que básicamente se divide en 4 fases: Análisis, Diseño, Codificación y Pruebas. Un conjunto de los modelos tradicionales de desarrollo proponen que ese esquema se ejecute de formal línea para la implementación de las aplicaciones.

En la realidad, la adaptación de estos esquemas secuenciales a la práctica es poco frecuente, ya que durante el desarrollo de una aplicación, tienden a aparecer nuevos requerimientos y/o imprevistos durante la ejecución de cualquiera de las fases, lo cual conlleva a la re planificación de las actividades, ejecutando nuevamente las 4 fases del método de desarrollo.

Se ha decidido entonces trabajar con la metodología de desarrollo XP [**14**] (eXtreme Programming). XP es un método de programación ágil [**15**] que se diferencia de las metodologías tradicionales principalmente porque pone más énfasis en la adaptabilidad que en la previsibilidad. XP considera que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una mejor aproximación y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo. Los valores de programación extrema que principalmente se explotan son el desarrollo iterativo e incremental, detección y corrección constante de errores antes de cada entrega a través de pruebas modulares y continuas. A continuación se describen los aspectos o fases en la cual XP hace énfasis a la hora de desarrollar un proyecto.

6.1.1 Análisis y Planificación

Durante la fase de análisis se definen los requerimientos, los cuales son desarrollados mediante iteraciones que cubren pequeñas funcionalidades o características requeridas. Parte de esta planificación consiste en organizar los requerimientos en función de su prioridad, de esta manera se tiene o se puede esbozar una visión general de las actividades que se pueden hacer de forma lineal o en paralelo.

6.1.2 Diseño

Una iteración en eXtreme Programming comienza con el diseño. Durante la fase de diseño de cada iteración se crea la estructura lógica del módulo a desarrollar. Los principales lineamientos de esta etapa son:

- Aplicar simplicidad al expresar las cosas solo una vez, sin añadir funcionalidades anticipadamente.
- Definir los métodos y las estructuras de datos que van a interactuar en un módulo.

- Organizar las ideas en estructuras lógicas de programación.
- Crear soluciones puntuales o programas sencillos para un problema específico que haya sido planteado dentro de los requerimientos.

6.1.3 Codificación

Una vez analizados los requerimientos y diseñada la "solución" para cubrir dichos requerimientos, se procede a codificar la misma. Es decir se implementan los métodos o funciones y las estructuras de datos previamente analizadas. Durante esta fase se documentan los aspectos más importantes por modulo. En eXtreme Programming la programación constituye la fase con mayor prioridad por sobre las demás actividades, ya que el objetivo es obtener resultados sustanciales al final de cada jornada. Los estándares relacionados a esta fase incluyen:

- Desarrollo del código basado en metáforas y estándares previamente acordados.
- Organización de un horario de trabajo semanal de unas 40 horas y cumplirlo a cabalidad, asegurando el trabajo óptimo pero sin sobrepasar las facultades mentales y físicas del equipo de desarrollo.
- Integración frecuente del código con los módulos ya probados y funcionales.

6.1.4 Pruebas

Toda iteración culmina con la verificación de la solución creada para asegurar que se cumpla con los requerimientos planteados al inicio. Al finalizar cada iteración, a través de pruebas unitarias se busca corregir cada nuevo desarrollo y de esta manera descartar errores puntuales y validar que todos los datos arrojados por el módulo sean correctos, es decir, se verifica que cada módulo tenga el comportamiento esperado.

6.2 Tecnologías a Utilizar

Será utilizado el lenguaje de programación C, por su robustez, documentación y por ser el lenguaje predilecto a la hora de programar sobre sistemas operativos GNU/Linux. GNU C Compiler (GCC) y GNU *make* para compilar el código fuente. La plataforma de virtualización *virtualBox* y *netkit* para emular otro(s) host(s) y/o interfaces de red. GNU Debugger (*gdb*) y *valgrind* para depuración del código. QT Creator como IDE de desarrollo, y finalmente *wireshark* y *tcpdump* para analizar el tráfico y/o tramas de red intercambiadas entre los hosts.

A continuación se describe brevemente las tecnologías a utilizar:

- **C**: Sin lugar a dudas el lenguaje de programación más famoso de la era moderna de la computación. En las manos de un conocedor y buen programador una herramienta muy poderosa y versátil. En manos inexpertas un récipe para el desastre. Vale la pena mencionar que la mayoría de los sistemas operativos han sido programados sobre este lenguaje (GNU/Linux incluido) así como la implementación original del protocolo TCP/IP hecha por la universidad de Berkeley para el sistema operativo BSD (y de la cual deriva todas, o casi todas, las implementaciones de TCP/IP modernas).
- **GCC**: El compilador C creado por el Grupo GNU. Robusto y eficiente. Prácticamente es el compilador C por defecto del sistema operativo GNU/Linux.
- **Make**: Utilidad creada por el grupo GNU para automatizar la compilación cuando de grandes proyectos se trata. Una herramienta muy poderosa y versátil, y en honor a la verdad algo compleja de entender (a profundidad).
- **VirtualBox**: Producto de virtualización orientado a estaciones de trabajo, desarrollado por Innotek GmbH, para arquitecturas x86 y AMD64/Intel64, de uso

empresarial y doméstico. Proporciona un ambiente de ejecución similar al de un computador físico con CPU, BIOS, tarjeta gráfica y de red, memoria RAM, disco duro, etc.

- **Netkit**: Un sistema operativo GNU/Linux (específicamente Debian) reducido (sin interfaz gráfica) corriendo como un proceso de usuario. *netkit* fue creado con la idea de simular ambientes de red a bajo (o ningún) costo. Permite crear interfaces de red virtuales y viene integrado con software (demonios) que permiten emular routers, switches y otros dispositivos de red. Una herramienta muy útil y versátil.
- **GNU Debugger (*gdb*)**: Depurador estándar para el sistema operativo GNU/Linux. Es portable y funciona para varios lenguajes de programación.
- **Valgrind:** Herramienta libre que ayuda en la depuración de problemas de memoria y rendimiento. Entre los problemas que *valgrind* ayuda a detectar se encuentran los siguientes:
    - Uso de memoria no inicializada.
    - Lectura/Escritura sobre memoria previamente liberada.
    - Lectura/Escritura fuera de los límites de memoria dinámicamente asignada.
    - Fugas de memoria.
- **QT Creator**: IDE (Integrated Developmente Environment) creado por Trolltech para el desarrollo de aplicaciones. Su “intellisense” o característica de autocompletado es realmente formidable.
- **Wireshark:** Antes conocido como “Ethereal”, es un analizador de protocolos utilizado para solucionar problemas en redes de comunicación, ampliamente empleado como herramienta didáctica para la educación. Posee una interfaz gráfica y muchas opciones de organización y filtrado de información. Permite ver todo el tráfico que pasa a través de una red, estableciendo la configuración de las interfaces de red en modo “promiscuo”.
- **Tcpdump:** Analizador de protocolos, similar a *wireshark* pero con una diferencia importante: funciona sobre línea de comandos (sin interfaz gráfica).

7. **Marco Aplicativo**

Este capítulo describirá en base a la metodología planteada en el Capítulo 6 las iteraciones tomadas en cuenta para la implementación de la solución, donde cada iteración muestra su proceso evolutivo en base a las fases de diseño, codificación y pruebas.

7.1 Análisis General

Tal como fue planteado anteriormente LLMNR es un servicio de resolución de nombres alternativo para GNU/Linux y este corre como un demonio dentro del mismo (como todo servicio en GNU/Linux). Inicialmente LLMNR no requiere de archivo de configuración para operar (puesto que es un protocolo del tipo “Zero Configuration” [**16**]), más sin embargo existen ciertos parámetros que pueden ser configurados en caso que el administrador y/o usuario así lo requiera. Tal como se puede apreciar en [**3**], LLMNR consta de un Sender y un Responder, que son sus 2 tareas básicas y principales. Aunque ambos están estrechamente relacionados, cada uno puede (y en el caso de GNU/Linux debe) operar de manera independiente.

Este análisis permitió de manera global determinar los principales requerimientos de la aplicación que permitiera cubrir los requerimientos de la aplicación definidos en el Capítulo 2. A continuación se muestra, a grandes rasgos, como fue planteada la lista de requerimientos para implementar el demonio:

- Diseño del archivo de configuración en texto plano.
- Diseño de estructuras para manipular mensajes LLMNR.
- Diseño de estructuras para manipular las interfaces de red.
- Diseño de estructuras para manipular los nombres autoritativos.
- Diseño de estructuras para almacenar la cache.
- Desarrollo de procedimientos que permite leer el archivo de configuración y los parámetros asociados.
- Desarrollo de procedimientos para el intercambio de mensajes LLMNR entre hosts utilizando UDP sockets.
- Desarrollo de procedimientos para el manejo de las interfaces de red.
- Desarrollo de procedimientos para implementar la verificación de unicidad de nombres y la detección y resolución de conflictos.
- Desarrollo de procedimientos para el manejo de señales.
- Desarrollo de procedimientos para el manejo de mensajes de loggin (Syslog).
- Desarrollo de procedimientos para implementar “caching”.

7.2 Desarrollo del Responder

El demonio LLMNR se divide en 2 partes, el Sender y el Responder, los cuales operan de manera independiente. Las iteraciones del 1 al 4 explican cómo se desarrolló el Responder. A continuación se especifican cada una de las iteraciones y las fases implementadas en ellas, según el modelo de programación XP.

7.2.1 Iteración 1: Diseño y Lectura del Archivo de Configuración

Este es un archivo de texto plano con el nombre de “llmnr.conf” ubicado en el directorio raíz de LLMNR. Este directorio raíz se encuentra en “/etc/llmnr”.

- **Fase de Diseño:** El archivo de configuración se divide en varias secciones, o parámetros de configuración. Cada parámetro es una “tupla” del tipo “Parámetro

valor1 valor2…valorN", donde la cantidad de valores depende del tipo de parámetro que se esté configurando. A continuación se especifican cada uno de los parámetros:

- **Sección hostname:** Para indicar el/los nombre(s) al cual el Responder será autoritativo y/o contestara mensajes LLMNR. Es importante destacar 3 cosas:
  - ✓ Tal como especifica el RFC 4795 [**3**], un host puede ser autoritativo de varios nombres o hostnames.
  - ✓ Un nombre o hostname debe ser del tipo FQDN (Fully Qualified Domain Name), es decir, tal como lo especifica el RFC 1034 [**1**] y el RFC 1123 [**17**] cada etiqueta debe contener máximo 63 caracteres[1], el tamaño total del nombre no puede exceder los 255 caracteres, solo son admitidos guiones (-) y caracteres alfanuméricos, y el nombre no puede empezar ni terminar con el carácter guion (-).
  - ✓ Si ningún nombre es provisto en el archivo de configuración "llmnr.conf" entonces el Responder tomará como nombre autoritativo el nombre de la máquina que le provea, el sistema. En caso de que la máquina pertenezca a un dominio el Responder adicionalmente tomará, tal como lo especifica el RFC 4795 [**3**], como nombre autoritativo la combinación (formando un FQDN) del nombre de la máquina con el dominio al cual esta pertenece.

  En la Figura 7.1 se muestra la sección hostname dentro de un archivo llmnr.conf de ejemplo.

```
# hostname config (a machine can have several hostnames):
hostname maquina1
hostname maquina1.com
hostname maquina1.com.ve
hostname otro-nombre
```

Figura 7.1: Parámetro hostname

- **Sección iface:** Para indicar las interfaces en las que el Responder escuchará peticiones LLMNR y en cual protocolo (IPv4 y/o IPv6). En este caso los valores posibles para esta etiqueta son "ipv4", "ipv6" o "dual" (IPv4 e IPv6). La Figura 7.2 muestra un ejemplo de la sección iface:

```
# interfaces that will listen to LLMNR petitions:
iface eth0 dual
iface eth1 ipv6
iface wlan0 ipv4
iface wlan0 ipv6
```

Figura 7.2: Parámetro Iface

[1] Un nombre de dominio está compuesto por una o más partes (etiquetas) concatenadas y separadas por un punto (.). Ejemplo: abc.com ("abc" y "com" son 2 etiquetas).

- **Sección log_xxx:** Donde “XXX” puede tomar los siguientes valores: “facility”, “level”, “conflicts” y “conflicts_on”. En esta sección se pueden definir básicamente 2 cosas distintas:
  - ✓ El “Facility y el “Level” para reportar errores a Syslog (el demonio de loggin de GNU/Linux). Los valores por defecto para este parámetro son: “LOG_DAEMON” y “LOG_ERR” respectivamente pero puede tomar cualquier valor posible referente al “facility” y el “level” que admite el demonio Syslog.
  - ✓ Si debe o no registrarse los conflictos de nombre detectados y en cual lugar debe hacerse. En caso de que no se especifique un lugar entonces se registrarán vía Syslog (LOG_DAEMON y LOG_WARNING). Para el parámetro “log_conflicts” los posibles valores son “yes” o “no”. La Figura 7.3 contiene un ejemplo de todos los parámetros configurables de la sección log_xxx:

```
# Syslog config for daemon errors:
log_facility LOG_DAEMON
log_level LOG_ERR

# Log LLMNR conflicts (default log is via Syslog):
log_conflicts yes
log_conflicts_on /var/tmp/log.file
```

Figura 7.3: Parámetro log_xxx

- **Sección Adicional:** Esta sección abarca los parámetros de configuración de los registros “adicionales” de DNS. Tal como especifica el RFC 4795 [**3**], los registros “A”, “AAAA” y “PTR” se derivan automáticamente. Mas sin embargo, un Responder LLMNR podría contestar cualquier registro válido DNS, como por ejemplo “MX”, “CNAME”, “NS”, etc. En este caso en particular, los registros adicionales que pueden configurarse son “MX” y “TXT”. La razón de por qué estos 2 solamente es debido a que el resto de los registros DNS (exceptuando “MX” y “TXT”) carecen de sentido bajo el ambiente en el que opera el protocolo LLMNR. Para el caso del registro “MX” los posibles valores que puede tomar es “Preference” (un número entero) y “Exchange” (un FQDN). Para “TXT” puede tomar cualquier texto contenido en una línea. La Figura 7.4 contiene un ejemplo de la sección adicional:

```
# Resource records config (only MX and TXT supported):
MX 10 mail.com.ve
TXT this is a txt record
```

Figura 7.4: Parámetros Adicionales

- **Fase de Codificación:** En la fase anterior se describió el diseño del archivo de configuración. La función *readConfigFile()* lee el archivo secuencialmente, línea por línea (ignorando los comentarios que se denotan con el símbolo “**#**”). Haciendo uso de la función *split()*, cada línea es dividida en “tokens” y estos son almacenados en una lista enlazada. Posteriormente la lista enlazada es procesada

por la función *parseConfigFile()*, la cual se encarga de verificar si la línea leída del archivo de configuración es válida. En particular valida lo siguiente:

- El primer elemento (o token) de la lista es una palabra de un alfabeto reservado de parámetros.
- El número de elementos (o tokens) es igual al número de elementos esperados para ese parámetro en particular.

Si todas las verificaciones son correctas entonces se invoca a la función "validXXX()" (donde XXX es el nombre del parámetro leído) correspondiente, la cual se encargara de verificar que el resto de los valores son válidos para el parámetro en cuestión.

- **Fase de Pruebas:** Estas pruebas consistieron en verificar que cada uno de los parámetros presentes en el archivo fuesen leídos correctamente, contemplando posible errores de sintaxis en la estructura del archivo, tales como símbolos no definidos, número de líneas máximo y tamaño total de línea permitido. También se verificó la correcta lectura de los comentarios. Es importante destacar que se hizo uso intensivo de la herramienta *valgrind* para detectar posibles errores (tales como fuga de memoria, referencias a memoria inválida o sin inicializar) en las listas enlazadas y estructuras que se emplean a la hora de leer y "parsear" el archivo de configuración. También es importante destacar que:
  - En aras de la simplicidad y la flexibilidad las secciones del archivo no están obligadas a estar en un orden específico.
  - Se maneja el caso en que no exista archivo de configuración. Dada esta situación, se crea un archivo de configuración de ejemplo y se toman valores por defecto en los parámetros requeridos para la correcta ejecución del Responder:
    - ✓ El nombre autoritativo del Responder se derivara automáticamente invocando la llamada de sistema *gethostname().*
    - ✓ Se escuchará peticiones LLMNR (IPv4 e IPv6) en todas las interfaces de red disponibles.
    - ✓ Se registrará errores del demonio vía Syslog en el "level" y "facility" previamente definidos.
    - ✓ Se registrará conflictos de nombre vía Syslog en el "level" y "facility" previamente definidos.

7.2.2 Iteración 2: Envió y Recepción de Mensajes LLMNR (Responder)

Haciendo uso de la biblioteca sockets que provee la librería GNU Libc, fueron desarrollados procedimientos para el intercambio de mensajes LLMNR entre PCs a nivel de capa de transporte según el modelo de referencia OSI.

- **Fase de Diseño:** En LLMNR existen 2 tipos de mensajes, LLMNR query y LLMNR answer. Estos mensajes viajan sobre el protocolo UDP (aunque también es válido usar el protocolo TCP). Los mensajes query y answer se muestran en la Figura 7.5 y Figura 7.6 respectivamente.

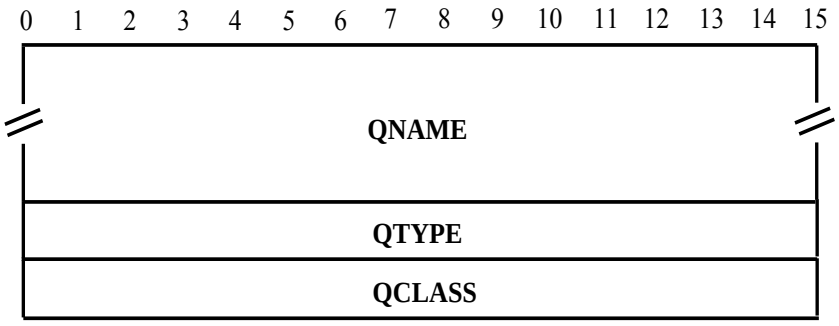


Figura 7.5: LLMNR Query

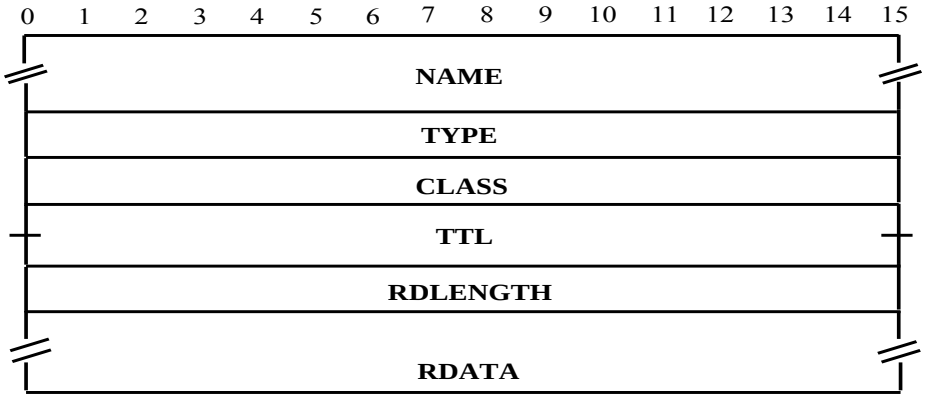


Figura 7.6: LLMNR Answer

El Responder creará un socket de recepción escuchando peticiones en el puerto designado para LLMNR (TCP/UDP 5355). Este socket escuchara todas los requests o peticiones (queries) LLMNR a través del hilo principal (en un bucle infinito) de la aplicación. Una vez arribado un mensaje, se procederá a recibir el mismo (el payload). Posteriormente se creará un hilo del tipo “desatendido” (este tipo de hilo es para indicarle al hilo principal que no debe esperar la culminación del hilo recién creado) al cual se le pasará el mensaje recién recibido más otra cantidad de parámetros necesarios. Este hilo se encargará de procesar dicho mensaje, primero verificando que sea un mensaje válido y en caso de que así sea, se encargará de crear y enviar la respuesta (answer) LLMNR a través del socket de recepción.

- **Fase de Codificación:** La comunicación entre PCs se hace a través de capa de transporte, a través del protocolo UDP, sin embargo es posible llevar a cabo dicha comunicación a través del protocolo TCP (ciertas condiciones aplican). Es por esto que es necesario crear, al menos, 2 sockets de recepción, uno para el protocolo UDP, otro para el protocolo TCP, y ambos escuchando en el puerto 5355. Este tipo de sockets requieren parámetros específicos en su creación mediante la función *socket()* del API (Application Programming Interface) de C. En la Figura 7.7 se muestra la función

```
int socket(int domain, int type, int protocol);
```

Figura 7.7: Función socket

Donde *domain* especifica el dominio de comunicación en el cual el socket será creado. Para el intercambio de paquetes en capa de transporte este parámetro puede ser o AF_INET (IPv4) o AF_INET6 (IPv6), *type* indica el tipo de socket, conectado (SOCK_STREAM) o no conectado (SOCK_DGRAM) y por último *protocol* indica el protocolo soportado para la familia de direcciones, en este caso será IPPROTO_TCP para sockets conectados (TCP) e IPPROTOCO_UDP para sockets no conectados (UDP).

Es preciso recalcar que un socket del tipo AF_INET6 es compatible con un socket del tipo AF_INET (al revés no ocurre), es decir, se puede recibir paquetes IPv4 con un socket del tipo AF_INET6, sin embargo ciertas operaciones adicionales sobre el socket son requeridas a la hora de enviar o recibir mensajes IPv4, por lo que se decidió crear 2 sockets UDP, uno para IPv4 (AF_INET) y otro para IPv6 (AF_INET6), ambos escuchando en el puerto LLMNR designado (5355). Para el caso del protocolo TCP se creó un solo socket del tipo AF_INET6, esto debido a que TCP no es el principal protocolo de intercambio de mensajes LLMNR, sino más bien su uso se limita a un protocolo secundario (cuando el tamaño del mensaje LLMNR es tan grande que no cabe en un datagrama UDP).

Debido a que los mensajes LLMNR query son enviados a una dirección multicast (224.0.0.252 y FF02::1:3 para IPv4 e IPv6 respectivamente), es necesario realizar una operación sobre los sockets de escucha (los 2 sockets UDP) que permita suscribirse a un grupo multicast (224.0.0.252 y FF02::1:3) para indicarle al sistema operativo que no debe descartar los mensajes enviados a las direcciones especificadas anteriormente (de lo contrario un mensaje dirigido a una dirección multicast de un grupo no suscrito seria descartado a nivel de capa de enlace).

Para la suscripción a los grupos multicast requeridos se usa la función descrita en la Figura 7.8.

```
int setsockopt(int sockfd, int level, int optname,
const void *optval, socklen_t optlen);
```

Figura 7.8: Función setsockopt

Donde *sockfd* es el valor devuelto por la función *socket()* especificada en la figura Figura 7.7, *level* es IPPROTO_IP para el caso del protocolo IPv4 y IPPROTO_IPV6 para el caso del protocolo IPv6, *optname* es IP_ADD_MEMBERSHIP e IPV6_ADD_MEMBERSHIP (para IPv4 e IPv6 respectivamente), *optval* es un apuntador *void* y *optlen* es el tamaño del tipo de dato al que apunta *optval*. Es preciso destacar 2 cosas:

- En C, los apuntadores del tipo *void* se usan, generalmente, cuando se quiere trabajar con distintos tipos de datos sin la necesidad de especificar exactamente cuál es el tipo de dato concreto.
- La función *setsockopt* se encarga de establecer las distintas opciones que GNU/Linux permite establecer en un socket (siendo una de ellas suscribirse a un determinado grupo multicast).

Entonces con el apuntador *void optval* y el entero *optlen, setsockopt* puede determinar cuál es el tipo de dato con el que se está trabajando y proceder en

consecuencia. En este caso, para IPv4, *optval* será una estructura descrita en la Figura 7.9, y *optlen* será el tamaño de la estructura.

```
struct ip_mreqn {
        struct in_addr imr_multiaddr;
        struct in_addr imr_address;
        int imr_ifindex;
};
```

Figura 7.9: Struct ip_mreqn

Donde *imr_multiaddr* es la dirección del grupo multicast a la que la aplicación quiere suscribirse (224.0.0.252), e *imr_address* es la dirección de la interfaz de red que quiere suscribirse al grupo, en este caso será la dirección IPv4 de cada interfaz de red que vaya a escuchar peticiones LLMNR. Finalmente *imr_ifindex* es el número de la interfaz que quiere suscribirse al grupo.

Para el caso de IPv6, *optval* será un apuntador a una estructura descrita en la Figura 7.10: Struct ipv6_mreqy *optlen* será el tamaño de la estructura.

```
struct ipv6_mreq {
        struct in6_addr ipv6mr_multiaddr;
        int ipv6mr_ifindex;
};
```

Figura 7.10: Struct ipv6_mreq

Donde *ipv6mr_multiaddr* es la dirección del grupo multicast a la que la aplicación quiere suscribirse (FF02::1:3), e ipv6mr es el número de la interfaz que quiere suscribirse al grupo, en este caso será el número de interfaz de cada interfaces (valga la redundancia) que quiera escuchar peticiones LLMNR.

Para la recepción de mensajes se usa la función descrita en la Figura 7.11.

```
ssize_t recvmsg(int s, struct msghdr *msg, int flags);
```

Figura 7.11: Función recvmsg

Generalmente, a la hora de programar sockets de tipo UDP, tiende a usarse la función *recvfrom()*. En este caso dicha función es insuficiente, ya que, una vez recibido un mensaje, es necesario conocer, entre otras cosas, por cual interfaz de red arribó dicho mensaje (para el caso de máquinas que tengan más de una interfaz de red). Esa información puede obtenerse de la siguiente manera:

- Estableciendo la opción IP_PKTINFO o IPV6_RECVPKTINFO (IPv4 e IPv6 respectivamente).
- Invocando la función *recvmsg().*

La información requerida será guardada y/o tratada a través de la estructura *msg* (del tipo *msghdr*). Esta información es conocida como “ancillary data”. No se profundiza más en este aspecto debido a que es un tema algo largo y

relativamente complejo de explicar, más sin embargo es importante recalcar que es una parte importante, y fundamental de la aplicación.

- **Fase de Pruebas:** Para las pruebas unitarias en el envío y recepción de mensajes LLMNR, se utilizó la herramienta de análisis de protocolos *wireshark* (descrita en la Sección 6.2), con la cual fue posible visualizar, verificar y depurar cualquier error a la hora del intercambio de mensajes LLMNR. Es importante recalcar que la herramienta *valgrind* (también descrita en la Sección 6.2) fue utilizada para validar el correcto uso de la memoria a la hora de procesar y/o generar los mensajes LLMNR.

  Se puede apreciar el intercambio de mensajes LLMNR entre 2 PCs, usando la pila de protocolos IPv4 e IPv6. Una máquina realiza un query preguntando por el (los) registro(s) A (IPv4) de “maquina1” vía IPv4 e IPv6 (Figura 7.12 y Figura 7.13). Posteriormente “maquina1” responde ambos mensajes con el registro requerido vía IPv4 e IPv6 (Figura 7.14 y Figura 7.15):

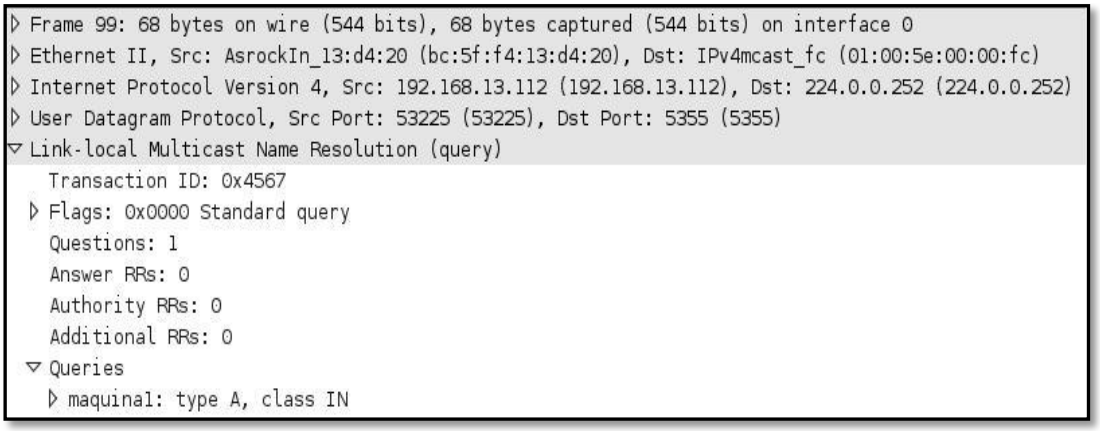


Figura 7.12: LLMNR query (IPv4)

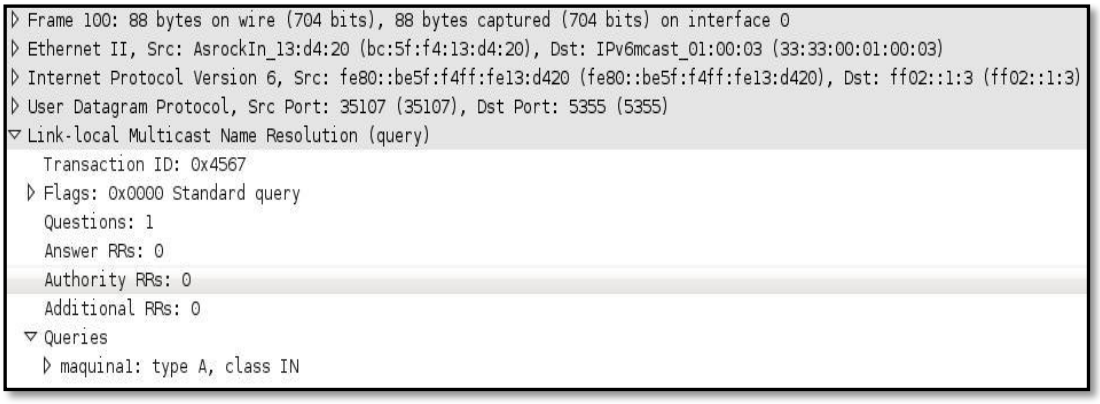


Figura 7.13: LLMNR query (IPv6)

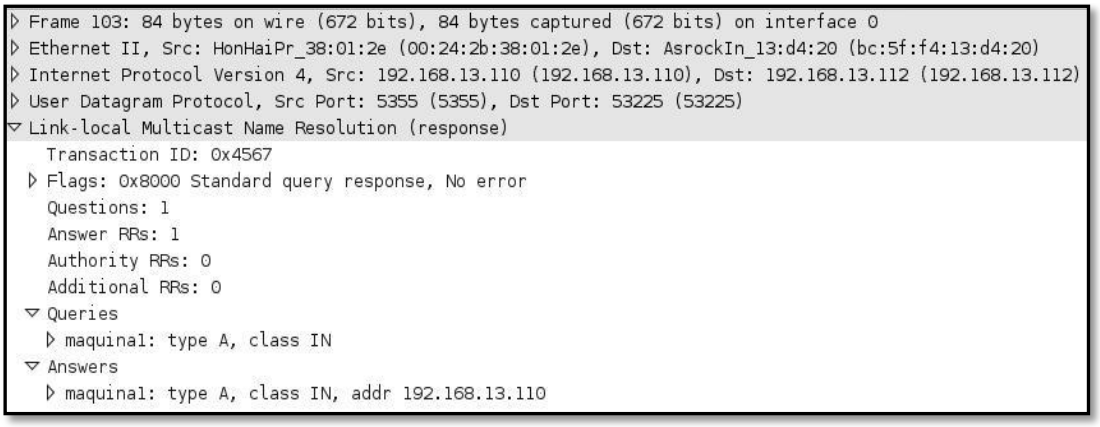


Figura 7.14: LLMNR answer (IPv4)

Figura 7.15: LLMNR answer (IPv6)

7.2.3 Iteración 3: Diseño y Manejo Dinámico de las Interfaces de Red

Tal como se especifica en el RFC 4795 [**3**], la implementación de un demonio LLMNR debe cumplir, entre muchas otras cosas, las siguientes 2 características:

- Derivar de manera automática los registros A, AAAA y PTR
- Hacer manejo dinámico de las interfaces de red y/o lidiar con los cambios que puede ocurrir en una interfaz de red. Estos cambios pueden ser:
  - Caída de una interfaz de red.
  - Subida de una interfaz de red.
  - Agregación de una dirección IPv4 o IPv6.
  - Supresión de una dirección IPv4 o IPv6.

Se puede apreciar entonces que las 2 características expuestas anteriormente se encuentran estrechamente ligadas, por lo que el manejo dinámico de las interfaces de red es una tarea crítica y medular en el diseño e implementación del demonio LLMNR.

- **Fase de Diseño:** Se utiliza una lista enlazada que llevará la información general de cada una de las interfaces de red configuradas para escuchar peticiones LLMNR. La estructura de las interfaces de red está compuesta de las siguientes variables:

- Nombre de la interfaz.
- Índice (identificador) de la interfaz.
- Una variable entera que representa un conjunto de banderas para llevar un registro de datos y/o estados asociados a la interfaz.
- Una lista (arreglo) de todas las interfaces de red “espejo” (interfaces que están en la misma red).
- Una lista enlazada de las direcciones IPv4 de la interfaz.
- Una lista enlazada de las direcciones IPv6 de la interfaz.

En la Figura 7.16 se puede apreciar la definición de la estructura.

```
typedef struct netIface {
        char *name;
        int flags;
        int ifIndex;
        U_SHORT mirroIfSz
        U_CHAR mirrorIfs[MAXIFACES]
        NETIFIPV4 *ipv4s;
        NETIFIPV6 *ipv6;
        struct netIface *next;
} NETIFACE;
```

Figura 7.16: Estructura de Interfaz de Red

Cabe destacar que U_SHORT es un dato de tipo *unsigned short* y U_CHAR es un dato de tipo *unsigned char*. NETIFIPV4 y NETIFIPV6 son estructuras que representa una dirección IPv4 e IPv6 respectivamente. En la Figura 7.17 y la Figura 7.18 puede apreciarse la definición de estas estructuras.

```
typedef struct netIPv4 {
        U_CHAR ipType;
        struct in_addr ip4Addr;
        U_CHAR ARECORD[ARECORDSZ];
        char PTR4RECORD[PTR4RECORDSZ];
        struct netIPv4 *next;
} NETIFIPV4;
```

Figura 7.17: Estructura NETIFIPV4

```
typedef struct netIPv6 {
        U_CHAR ipType;
        struct in6_addr ip6Addr;
        U_CHAR AAAARECORD[AAAARECORDSZ];
        char PTR6RECORD[PTR6RECORDSZ];
        struct netIPv6 *next;
} NETIFIPV6;
```

Figura 7.18: Estructura NETIFIPV6

Es preciso comentar las variables asociadas a las estructuras mencionadas previamente:

- **ipType:** Indica qué tipo de dirección IP se almacena. Para IPv4 este valor siempre será el mismo (IPV4IP). Para IPv6 dependerá de que tipo de dirección IPv6 se está almacenando (link-local, global, unique-local, etc).
- **ip4Addr/ip6Addr:** La dirección IPv4/IPv6 per se.
- **ARECORD/ AAAARECORD:** Contendrá una construcción (o derivación) parcial de un registro (respuesta LLMNR) A/AAAA (ver Figura 7.6).
- **PTR4RECORD/PTR6RECORD:** Contendrá la construcción del nombre PTR asociado a la dirección IP.

Para detectar cambios en las interfaces de red, se usa *NETLINK*. Este es un protocolo, orientado a datagramas, que se usa para comunicar información entre el kernel del sistema operativo GNU/Linux y los procesos corriendo en el espacio de usuario. Aunque el protocolo NETLINK está definido en el RFC 3549 [**18**], su uso está limitado única y exclusivamente (hasta el día de hoy) al Sistema Operativo GNU/Linux. En la Figura 7.19 se muestra la cabecera de un mensaje *NETLINK.*

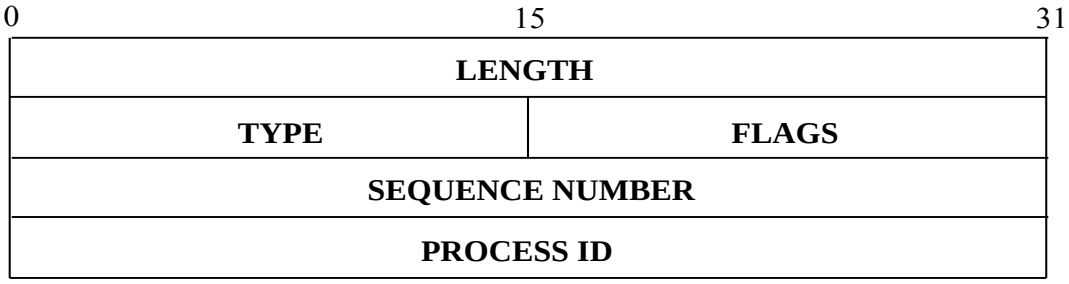


Figura 7.19: Cabecera NETLINK

A continuación se describe brevemente cada uno de los campos de la cabecera *NETLINK*:

- **LENGTH:** Tamaño del mensaje (incluida la cabecera)
- **TYPE:** El tipo de mensaje
- **FLAGS:** Banderas adicionales
- **SEQUENCE NUMBER:** El número de secuencia del mensaje
- **PROCESS ID:** ID del proceso que envía el mensaje.

Para recibir mensajes *NETLINK* que contengan información sobre cambios en las interfaces de red del sistema, es necesario suscribirse a un grupo multicast específico. Una vez hecho esto, cuando el sistema operativo tenga que informar sobre algún acontecimiento en el grupo multicast suscrito, la aplicación recibirá un mensaje. Es pertinente comentar que existe una cantidad diversa de grupos multicast a la cual una aplicación puede suscribirse, y que para cada grupo multicast existen distintos tipos (**TYPE)** de mensajes. Más específicamente, para el caso que nos atañe, cuando se genere un acontecimiento sobre cualquiera de las interfaces de red del sistema, *NETLINK* enviará un mensaje multicast, informando sobre el tipo de acontecimiento, y sobre cual interfaz se generó.

Previamente se mencionó una variable entera que representa un conjunto de banderas (flags) o estados y datos asociados a una interfaz dada. Dichas banderas están definidas con un tipo de dato *enum* o enumerado. En la Figura 7.20 se muestra la definición de cada una de las banderas.

```
enum IF_FLAGS {
        _IFF_NOIF
        _IFF_STATIC
        _IFF_DYNAMIC
        _IFF_LOOPBACK
        _IFF_RUNNING
        _IFF_INET
        _IFF_INET6
        _IFF_MCAST4
        _IFF_MCAST6
        _IFF_STATICINET
        _IFF_STATICINET6
        _IFF_CONFLICT
        _IFF_CDAR
};
```

Figura 7.20: Interface Flags

A continuación se describe cada una de ellas:

- **_IFF_NOIF:** No interface. El nodo es cabeza de lista y no una interfaz.
- **_IFF_STATIC:** Indica si la lista de interfaces es estática (configuradas vía el archivo */etc/llmnr/llmnr.conf*.
- **_IFF_DYNAMIC:** Indica si la lista de interfaces es dinámica (se escuchará en todas las interfaces de red disponibles en el sistema)
- **_IFF_LOOPBACK:** Indica si la interfaz es una interfaz loopback.
- **_IFF_RUNNING:** Indica si la interfaz está operativa o no.
- **_IFF_INET:** Indica si la interfaz tiene dirección IPv4.
- **_IFF_INET:** Indica si la interfaz tiene dirección IPv6.
- **_IFF_MCAST4:** Indica si la interfaz está suscrita al grupo multicast 224.0.0.252.
- **_IFF_MCAST6:** Indica si la interfaz está suscrita al grupo multicast FF02::1:3.
- **_IFF_STATICINET:** Indica si la interfaz debe operar en el protocolo IPv4 (configurada vía el archivo */etc/llmnr/llmnr.conf*).
- **_IFF_STATICINET6:** Indica si la interfaz debe operar en el protocolo IPv6 (configurada vía el archivo */etc/llmnr/llmnr.conf*).
- **_IFF_CONFLICT:** Indica si la interfaz es interfaz “espejo”, es decir, opera en la misma red que opera al menos otra interfaz de la máquina.
- **_IFF_CDAR:** Indica si la interfaz realizó el proceso de verificación de unicidad de nombre (descrito en la Sección 4.6).

- **Fase de Codificación:** La manipulación de la lista de interfaces se hace a través del conjunto de primitivas *NetIfListHead(), addNetIfNode(), remNetIfNode(), getNetIfNodeByIndex(), getNetIfNodeByName()*, con las cuales es posible crear, agregar, y encontrar interfaces en la lista de interfaces. La manipulación de las direcciones IP se realiza con las primitivas *addNetIfIPv4(), addNetIfIPv6(), remNetIfIPv4(), remNetIfIPv6(), getNetIfIpv4Node (), getNetIfIpv6Node(), con l*as cuales es posible agregar, remover o encontrar direcciones IP dentro de la lista de interfaces.

Cuando el demonio inicia, crea la lista de interfaces, en este caso sería una lista con un solo nodo, la cabeza, el cual es marcado como tal. Posteriormente lee el archivo de configuración, si existe alguna configuración para las interfaces entonces agrega las interfaces especificadas en el archivo con el protocolo especificado, en caso contrario, agregará a la lista todas las interfaces de red disponibles en el sistema, con los protocolos IPv4 e IPv6.

Es importante notar la diferencia entre operar con interfaces configuradas vía archivo de configuración (*/etc/llmnr/llmnr.conf*) y operar con interfaces sin configuración. En el primer caso, el administrador podría optar, por ejemplo, escuchar peticiones LLMNR en solo una interfaz dada (de entre varias que podría tener un equipo) en un protocolo específico (IPv4 o IPv6). Bajo este escenario, si una interfaz cambia, dicho cambio no necesariamente afecta la operación del demonio puesto que el cambio pudo darse en una interfaz o protocolo distinto al configurado por el administrador. Por lo tanto, con la ayuda de las banderas _IFF_STATIC, _IFF_STATICINET4 e _IFF_STATICINET6 se puede saber, sin necesidad de releer el archivo de configuración, si el hipotético cambio en una interfaz de red afecta la operación del demonio o no. En el segundo caso, puesto que no hay ninguna configuración especifica de interfaz, se escuchará por defecto, peticiones LLMNR en todas las interfaces disponibles, bajo ambos protocolos (IPv4 e IPv6), por lo que cualquier cambio en alguna interfaz de red si afectará la operación del demonio.

Una vez creada la lista de interfaces, se procede a llenar las direcciones IPv4 (si aplica) o IPv6 (si aplica). Las direcciones IPv4 o IPv6 se obtiene a través de la llamada a sistema descrita en la Figura 7.21.

```
int getifaddrs(struct ifaddrs **ifap);
```

Figura 7.21: Función getifaddrs

La definición del struct ifaddrs se muestra en la Figura 7.22.

```
struct ifaddrs {
        struct ifaddrs *ifa_next;
        char *ifa_name;
        unsigned int ifa_flags;
        struct sockaddr *ifa_addr;
        struct sockaddr *ifa_netmask;
        union {
                struct sockaddr *ifu_broaddr;
                struct sockaddr *ifu_dstaddr;
        } ifa_ifu;
};
```

Figura 7.22: Struct ifaddrs

La función *getifaddrs* retorna una lista enlazada de struct ifaddrs, donde cada nodo contiene cada dirección IP perteneciente a cada interfaz de red del sistema. Una vez obtenida esta lista, se invoca la función *fillInterfaces()*, la cual se encarga de

llenar, con el apoyo de las primitivas mencionadas anteriormente, las direcciones IP correspondientes en la lista de interfaces LLMNR.

Posteriormente con las funciones *fillARecord()*, *fillPtrRecord()* y *fillAAAARecord()* se derivan (y almacenan), de manera parcial, los registros A, AAAA y PTR correspondientes a cada interfaz. La derivación es parcial ya que el registro no se completa, si observamos la Figura 7.6, vemos que un registro (o respuesta) consiste de los siguientes campos:

- Name.
- Type.
- Class.
- TTL
- RDLENGTH.
- RDATA.

En este punto del programa todos estos son valores conocidos (incluso el TTL), por lo que es posible derivar el registro completo. Mas sin embargo, el nombre no se adjunta porque como se dijo antes, un Responder puede ser autoritativo de varios nombres, lo que implicaría tener registros duplicados por cada nombre, en su lugar, el nombre se adjunta al registro parcial (para finalmente formar un registro completo) a la hora de generar una respuesta.

Una vez completada las tareas descritas anteriormente, con la llamada a la función *joinMcastGroup()* se suscribirá la interfaz al grupo multicast LLMNR (224.0.0.252 o FF02::1:3) según aplique.

Para hacer uso del protocolo *NETLINK,* es requerido crear un socket de comunicaciones. Este se crea invocando a la función socket (mostrada en la Figura 7.7), tal como se muestra en la Figura 7.23

```
int socket(AF_NETLINK,SOCK_RAW,NETLINK_ROUTE);
```

Figura 7.23: Creación del Socket Netlink

Donde *domain es* AF_NETLINK, type es SOCK_RAW (SOCK_DGRAM también funciona) y *protocol* es NETLINK_ROUTE. En este caso *protocol* no hace referencia a un protocolo en sí, más bien, hace referencia a un módulo o sub-sistema del sistema operativo. La cantidad de sub-sistemas disponibles para NETLINK es amplia y variada, en este caso el sub-sistema de interés es NETLINK_ROUTE, el cual puede usarse para controlar los siguientes elementos del sistema operativo:

- Tablas de enrutamiento.
- Direcciones IP.
- Parámetros de configuración de los enlaces de red.
- Configuración de vecinos (neighbor discovery).
- Algoritmo de planificación para el manejo de colas en los paquetes de red.
- Clasificador de paquetes de red.

Este socket se usará para recibir los mensajes que genere el Kernel para el proceso de usuario. Para poder empezar a recibir mensajes primero hace falta suscribirse a un grupo multicast, para indicarle al Kernel que tipo de mensajes le interesa a la aplicación. En este caso, son 2 los grupos multicast de interés:

- RTMGRP_IPV4_IFADDR
- RTMGRP_IPV6_IFADDR

Para realizar la suscripción, se invoca la función *bind()* (Figura 7.24)

```
int bind(int sockfd, struct sockaddr *myaddr, socklen_t addrlen);
```

Figura 7.24: Función bind

Donde *sockfd* es el valor devuelto por la función *socket*, *myaddr* es un struct del tipo *sockaddr_nl* (descrita en la Figura 7.25), y *addrlen* es el tamaño del struct *sockaddr_nl*.

```
struct sockaddr_nl {
        sa_family_t nl_family
        unsigned_short nl_pad /* Siempre cero */
        pid_t nl_pid;
        __u32 nl_groups;
};
```

Figura 7.25: Struct sockaddr_nl

Donde *nl_family* siempre será AF_NETLINK, *nl_pid* es el *pid* (program id) de la aplicación que quiere suscribirse y *nl_groups* indica el, o los grupos, a los que la aplicación quiere suscribirse.

Para monitorear el socket NETLINK, y al igual que los otros sockets mencionados en la Sección 7.2.2, se usa la llamada a sistema *poll()*. Una vez recibido un mensaje, este se procesa con las funciones *handleNetlinkQuery()* y *getRta()*, las cuales se encargarán, con el apoyo de otras funciones, de tomar las acciones correspondientes. Es preciso destacar lo siguiente, cuando una interfaz de red cambia, digamos por ejemplo, se agrega una nueva interfaz de red al sistema (supongamos que esta interfaz levanta con 2 direcciones: una IPv4 y otra IPv6), se recibirá, vía socket NETLINK, al menos 2 mensajes:

- Un primer mensaje informando que se agregó una nueva dirección IPv4 a la interfaz.
- Un segundo mensaje informando que se agregó una nueva dirección IPv6 a la interfaz.

En ambos casos, el mensaje contendrá, por supuesto, la dirección IP que se agrega, el identificador de la interfaz que está agregando las direcciones IP antes mencionadas, así como un conjunto de banderas del dispositivo (*NETDEVICE)*.

Es importante destacar que la función *handleNetlinkQuery()* corre en el hilo principal de la aplicación y que dicha función no puede ser ejecutada si existe algún otro hilo corriendo dentro de la aplicación (típicamente, este otro hilo sería un query siendo atendido). Esto es así porque es una misión crítica garantizar la coherencia en la lista de interfaces. Potencialmente, la invocación a la función *handleNetlinkQuery()* podría alterar la lista (borrar una IP por ejemplo o establecer un apuntador a NULL), lo que podría derivar en un desastre (*segmentation fault* por ejemplo) si no se tienen los cuidados necesarios.

Para garantizar que no hay hilos corriendo y es seguro correr la función previamente mencionada se usa una variable global llamada *CountMutex*, la cual llevará la cuenta de los hilos corriendo. Por supuesto esta variable es accedida y modificada a través de un Mutex. Si hay un hilo corriendo entonces se espera *LLMNR_TIMEOUT* milisegundos y se intenta de nuevo.

- **Fase de Pruebas:** para verificar la "correctitud" en la construcción y manejo de la lista enlazada de interfaces, se efectuó una depuración a fondo de cada una de las primitivas y estructuras asociadas. Para la depuración se hizo uso intensivo de la herramienta *valgrind*, la cual es extremadamente efectiva a la hora de depurar listas enlazadas, y en general, el manejo de memoria en C, puesto que permite detectar errores de memoria no liberada, referencia a memoria no inicializada o referencia a memoria liberada. Cada una de las primitivas asociadas al manejo de la lista de interfaces fueron evaluadas con pruebas unitarias.

  Para verificar el correcto funcionamiento de NETLINK, se montaron diversos y números escenarios de prueba, que permitieron determinar los grupos multicast a los cuales la aplicación tenía que suscribirse, así como procesar, de manera correcta, los mensajes una vez recibidos. No está de más comentar que, aunque NETLINK es un protocolo sumamente útil, su documentación es escaza y limitada.

7.2.4 Iteración 4: Implementación de CDAR (Conflict Detection and Resolution)

La resolución (y detección) de conflictos y la verificación de unicidad de nombres es un paso importante y crucial del protocolo LLMNR. En el contexto LLMNR, verificar y defender un nombre, aunque teóricamente son 2 funcionalidades distintas en la práctica son la misma cosa. Para ambos casos, consiste en enviar un query LLMNR por el nombre que se quiere comprobar/defender, para validar o descartar si existe algún otro equipo en la red que conteste o sea autoritativo del mismo.

- **Fase de Diseño:** Cuando se opera en un equipo con varias interfaces de red, es importante tener en cuenta las siguientes consideraciones/situaciones:
  - Cada nombre debe ser verificado (o defendido cuando aplique) en cada una de las interfaces.
  - Si un nombre falla en la defensa en una interfaz dada, ese nombre dejara de ser autoritativo en esa interfaz solamente.
  - Si se opera sobre un equipo con interfaces "espejo" (2 o más interfaces operando en la misma red).

  Se utilizara una lista enlazada de nombres del tipo *NAME*.
  Esta lista contendrá cada uno de los nombres autoritativos (o potencialmente autoritativos) del demonio.

  En la

  Figura 7.26 se puede apreciar la definición de la estructura que contiene cada nombre.

```
typedef struct nameNode {
        char *name;
        char nameStatus;
        int ptrSz;
        U_CHAR ptr;
        U_SHORT authOnSz;
        U_SHORT notAuthOnSz;
        U_CHAR authOn[MAXIFACES];
        U_CHAR notAuthOn[MAXIFACES];
        Struct nameNode *next;
} NAME;
```

Figura 7.26: Struct nameNode

A continuación se describe el uso de cada variable:

- **Name:** Contendrá el nombre (potencialmente) autoritativo. Es preciso destacar que este nombre se guarda en formato “DNS” (donde cada etiqueta del nombre es precedido por el tamaño de la misma).
- **nameStatus:** Contendrá el estado en el que se encuentra el nombre (TENTATIVE u OWNER).
- **ptrSz:** Tamaño del registro *ptr*.
- **ptr:** Contendrá una derivación parcial del registro o respuesta a consultar del tipo PTR (cuando la consulta es del tipo PTR la respuesta no es una dirección IP, es un nombre).
- **authOnSz/notAuthOnSz:** Indica el número de elementos contenidos en el arreglo *authOn/notAuthOn* respectivamente.
- **authOn:** Contendrá el identificador (index) de todas las interfaces que pueden responder queries a este nombre (*name)*.
- **notAuthOn:** Contendrá el identificador (index) de todas las interfaces que no pueden responder queries a este nombre (*name).*

Existen 2 listas (arreglos) de interfaces, las que pueden responder queries y las que no. Parece algo redundante, y de hecho lo es. La razón para esta redundancia se basa en la ambigüedad que se generá a la hora de defender (o verificar) un nombre en una máquina que contiene interfaces “espejo”. Supóngase que se tiene una máquina con 2 interfaces, eth0 y wlan0, ambas conectadas a la misma red (“espejo”). En el momento en que a la hipotética máquina le toque defender (o verificar) el nombre, podría darse el caso (asumiendo que existe otra máquina en la red que responda al mismo nombre) que una interfaz (wlan0 por ejemplo) pierda la defensa y la otra interfaz (eth0) gane. En ese escenario una interfaz (eth0) podrá contestar queries al nombre mientras que la otra (wlan0) no podrá. Esto generá una situación ambigua, puesto que al ganar la interfaz eth0 el nombre, el mismo es propiedad de la máquina en esa red, sin embargo una de las interfaces de la máquina (wlan0) no puede responder a pesar de estar conectada a la red donde la máquina es propietaria del nombre, aunado al hecho que si eth0 por alguna razón cae, el nombre quedaría huérfano (estando wlan0 aun activa).

El RFC 4795 no precisa este escenario, por lo que el comportamiento en dicha situación queda a decisión de la implementación, y es por esto que se utilizan las 2

listas (arreglos). Cuando una interfaz (espejo) pierde una defensa (o verificación), la perdida se considera potencial puesto que antes de colocarse en la lista de interfaces no autoritativas del nombre, se buscará primero, si existe dentro de la lista de interfaces autoritativas del nombre alguna interfaz espejo, en cuyo caso se considera el conflicto ganado (podría llamarse a esto “tráfico de influencias”). En caso contrario, cuando una interfaz gana un conflicto o lleva a cabo exitosamente una verificación del nombre, se colocará a esta dentro de la lista de interfaces autoritativas del nombre, posteriormente se verificará si existe dentro de la lista de interfaces no autoritativas alguna interfaz espejo.

Cuando la aplicación (Responder) recibe un mensaje LLMNR que alerta un conflicto, dicha alerta se almacena en una lista de conflictos pendientes de resolver. Los conflictos en esta lista se atenderán uno a la vez (solo habrá un hilo corriendo para todos los conflictos) y bajo el orden FIFO (First in First out). Posteriormente cada conflicto será guardado (tal como lo especifica el RFC 4795) en el lugar especificado por el archivo de configuración. En la Figura 7.27 se muestra la estructura contentiva de conflictos.

```
typedef struct __conflict {
        NAME *cName;
        int type;
        int family;
        char *logPath;
        void *cPeerIp;
        int ifIndex;
        struct tm cTime;
        struct __conflict *next;
} CONFLICT;
```

Figura 7.27: Struct conflict

A continuación se explica cada una de las variables:

- **cName:** Contendrá el nombre en conflicto.
- **type:** El tipo de registro en conflicto (A, AAA, etc)
- **family:** La familia de direcciones del mensaje que reportó el conflicto (AF_INET o AF_INET6).
- **logPath:** Ruta donde registrar el conflicto.
- **cPeerIp:** Dirección IP del peer que reporto el conflicto.
- **ifIndex:** Identificador de la interfaz que recibió la alerta de conflicto.
- **cTime:** Hora y fecha en la cual se recibió el conflicto (en formato local time).

- **Fase de Codificación:** La función *invokeCdar()* es llamada desde el hilo principal de la aplicación y se encarga de discriminar el tipo de llamada que debe usarse para invocar el proceso de resolución y detección de conflictos (CDAR). En la Figura 7.28 se muestra el prototipo de la misma

```
void invokeCdar(U_CHAR type, int ifIndex, NAME *name);
```

Figura 7.28: Función invokeCdar

Donde *name* es el nombre que se va a verificar/defender, *ifIndex* es la interfaz (o sub-red) donde se va a verificar/defender y *type* es el tipo o estado de invocación. Existen 3 estados distintos:

- **CDARINIT:** Cuando el demonio arranca. Se defienden todos los nombres en todas las interfaces.
- **CDARIFACEUP:** Cuando se agrega (sube) una interfaz de red. Se defienden todos los nombres en la interfaz que recién subió.
- **CDARCONFLICT:** Cuando se recibe un mensaje de alerta de conflicto. Se defiende el nombre en conflicto en la interfaz que recibió el mensaje de alerta.

CDARINIT solo ocurre una vez mientras que los otros 2 tipos pueden ocurrir n veces. Cada vez que se convoca un proceso *cdar*, se crea un hilo que se encarga de atender dicho proceso. *invokeCdar()* establece una variable global llamada *RunningCdar*, posteriormente crea el hilo que correrá el proceso. Una vez finaliza el hilo este se encargará de establecer de nuevo a 0 la variable *RunningCdar. Con esto se* controla que solo exista una instancia de *cdar* corriendo. Esto es para garantizar que los estados *cdar* sean mutuamente excluyentes debido a que CDARIFACEUP es invocado por un evento NETLINK.

Cuando el proceso *cdar* arranca, se invoca a la función *cDar().* El prototipo de esta se muestra en la Figura 7.29*.*

```
void cDar(NAME *name, NETIFACE *iface, NETIFACE *ifs);
```

Figura 7.29: Función cdar

Como se dijo anteriormente *name* es el nombre a ser defendido/verificado e iface la interfaz o subred donde se dará el evento. *ifs* es la lista de todas las interfaces de red LLMNR. La función *cdar()* creará un paquete LLMNR, el cual es un query por el nombre a verificar/defender del tipo ANY. Este paquete será enviado a las direcciones multicast 224.0.0.252 y FF02::1:3 (nótese que en realidad se están generando 2 paquetes). Para esto se crearan 2 sockets invocando las funciones *createC4Sock()* y *createC6Sock()*. Estas funciones crearán un socket, uno IPv4 y otro IPv6 respectivamente. Haciendo uso de la función *setsockopt()* (mostrada en la Figura 7.8) y las opciones *IPPROTO_IP*, *IP_MULTICAST_IF* (para IPv4) e *IPPROTO_IPV6, IPV6_MULTICAST_IF* (para IPv6) se puede establecer por cual interfaz de red saldrá el paquete, en este caso se usará como interfaz de salida la interfaz especificada por *iface*.

Posterior al envío se invoca a la función *recvMsg()*, la cual esperara *LLMNR_TIMEOUT* milisegundos, y luego verificará si alguna respuesta llegó. Es importante destacar que 2 paquetes son enviados (uno IPv4 y otro IPv6), por lo que es posible recibir respuestas en ambos protocolos. En este punto es obligatorio seleccionar cuál de los 2 protocolos se va a tomar para sondear posible respuestas (la razón de por qué se envían 2 paquetes es un tema de redundancia, si un query IPv4 falla quizás uno IPv6 tenga éxito, o viceversa). En este caso el macro *PRIORITY_IPV4 se usa* para discriminar cual es el camino a tomar. En la Figura 7.30 se muestra parte del código de selección de la función *recvMsg()*.

```
#ifdef PRIORITY_IPV4
        if (recvMsg4(fd4,params))
                return SUCCESS;
        else if (recvMsg6(fd6,params))
                return SUCCESS;
#else
        if (recvMsg6(fd6,params))
                Return SUCCESS;
        else if (recvMsg4(fd4,params))
                return SUCCESS;
#endif
```

Figura 7.30: Código de Selección

Se puede apreciar que cuando el macro *PRIORITY_IPV4* está definido, se invoca a la función *recvMsg4()*, la cual buscará posibles respuestas IPv4 al query previamente enviado. Si se recibe una respuesta entonces se retorna, en caso contrario (falló IPv4) se invoca a *recvMsg6()* para sondear si existe alguna respuesta vía IPv6. Si el macro *PRIORITY_IPV4* no está definido entonces se busca primero respuestas vía IPv6 y si falla entonces IPv4. Si ninguna respuesta es recibida entonces se vuelva a realizar el proceso hasta ahora descrito. El número de intentos está definido por el macro *QUERYMAXTRIES.* El RFC 4795 recomienda que se realice máximo 3 intentos antes de considerar que no existe respuesta.

Ya dentro de recvMsg4 o recvMsg6 (según sea el caso), para colectar todas las posibles respuestas se invoca a la llamada a sistema *recvmsg()* (Figura 7.11) en un bucle infinito con la bandera *MSG_DONTWAIT activada*, la cual hace que la llamada retorne de inmediato (no bloqueante). Si el socket no recibió respuesta entonces el sistema establecerá la variable global de error *errno* con el valor *EWOULDBLOCK,* en cuyo caso se detiene el bucle y la búsqueda de respuestas. Si por el contrario, una respuesta es recibida entonces se invoca a una función auxiliar que se encargará de verificar las siguientes cosas:

- **Correctitud del Mensaje:** Si la cabecera es válida, si el id de respuesta coincide con el id del query, si la respuesta tiene el bit T (Tentative) encendido o si el mensaje es una respuesta al nombre hecho en la consulta.
- **Auto Respuesta:** Si el mensaje vino de alguna de mis interfaces. Para esto se compara la dirección IP origen en la respuesta contra la IP de todas las interfaces (excepto la interfaz que recibió la respuesta). Si existe algún match entonces la respuesta vino de la misma máquina, en cuyo caso se ha detectado una interfaz espejo y ambas (la que recibió la respuesta y la que lo originó) son marcadas como interfaces “espejo”. Para marcarlas se agrega o se levanta la bandera _IFF_CONFLICT.
- **Ganador/Perdedor:** Finalmente, si es una respuesta válida (tentativa no auto respuesta) entonces se trata de un conflicto, en cuyo caso se realiza la comparación lexicográfica de la IP origen y destino en la respuesta. Si se pierde la comparación lexicográfica entonces inmediatamente se marca la interfaz de salida como no autoritativa para el nombre que está siendo verificado/defendido.

En caso de que se gane la comparación lexicográfica, no puede marcarse el nombre como autoritativo o defendido (recordemos que se está en el proceso de verificar un nombre o atender un conflicto) hasta que se haya procesado cada una de las respuestas (es decir hasta que errno == EWOULDBLOCK). Si finalmente se prevalece entonces puede considerarse el nombre como autoritativo y se marca como tal, es decir se agrega el índice de la interfaz en la lista de autoritativos de *name*.

- **Fase de Pruebas:** Para verificar la correctitud del proceso de detección y resolución de conflictos, se efectuó una depuración minuciosa sobre las funciones que forman parte del proceso, especialmente aquellas que se encargan de recibir y sondear las posibles respuestas, esto por supuesto, con la ayuda de herramientas de programación tan útiles como *valgrind* y *gdb*. También se montaron diversos y múltiples escenarios de pruebas que permitieron validar que las correctas acciones fueron ejecutadas ante la variedad de posibilidades que se pueden presentar en una red a la hora de defender o verificar un nombre.

7.3 Desarrollo del Sender

El Sender es el que se encarga de realizar un query con el fin de resolver un nombre. Es importante destacar que para este desarrollo se reutilizó parte del código del Responder, más específicamente, a grandes rasgo el Sender no es más que el proceso *cdar* descrito en la Sección 7.2.4. Por supuesto tiene algunas diferencias con esta última, aparte el Sender incorpora procesos que no se dan en el Responder y que serán mencionados más adelante. Antes de comenzar a hablar de las iteraciones, existen 2 aspectos del Sender que no amerita una iteración más sin embargo es importante recalcar y se mencionan a continuación:

- Aunque de manera muy limitada, el Sender hace uso del archivo de configuración /etc/llmnr/llmnr.conf. La manera como se lee y se procesa el archivo de configuración es similar a las acciones realizadas por el Responder (descrito en la Sección 7.2.1). Existe una pequeña sección dentro del archivo que sirve para indicarle al Sender en cuales interfaces (en caso que así lo requiere el administrador) enviar queries a la hora de intentar resolver un nombre. Esta sección es identificada con la palabra reservada *queries_on* seguido del nombre de la interfaz. En la Figura 7.31 se puede ver un ejemplo de la sección.

```
# where to send LLMNR queries
queries_on eth0
queries_on wlan0
```

Figura 7.31: Parámetro queries_on

- El Sender hace uso de las interfaces de red descritas en parte de la Sección 7.2.3. La manera como la lista de interfaces es creada y manejada es similar a las acciones llevadas a cabo por el Responder, con la salvedad que en el Sender no se hace uso dinámico de las interfaces ya que no aplica.

7.3.1 Iteración 1: Queries y Respuestas

Esta iteración se encarga de generar el query LLMNR para intentar resolver el nombre que se quiere, y de recibir y almacenar las respuestas recibidas (si alguna). Funciona de

manera similar al proceso descrito en la Sección 7.2.4, pero con ciertas diferencias notables.

- **Fase de Diseño:** Para intentar resolver el nombre es preciso realizar un query en todas las interfaces disponibles, hasta que se obtenga una respuesta o se haya probado todas las interfaces sin obtener ninguna respuesta.

  A diferencia del Responder, el Sender tiene la tarea de detectar conflictos y alertar sobre los mismos cuando estos ocurran. Un conflicto se detecta cuando se recibe 2 o más respuestas provenientes de distintos orígenes. Por esta razón es de vital importancia almacenar todas las respuestas recibidas. En este caso, las respuestas serán almacenadas en una lista enlazada, donde cada nodo de la lista representa una respuesta. En la Figura 7.32 se muestra la estructura de cada respuesta.

```
Typedef  struct resps {
        void *ip;
        U_CHAR *buf;
        U_CHAR cBit;
        U_SHORT id;
        size_t ipLen;
        U_SHORT bufLen;
        U_SHORT anCount;
        int fromIface;
        struct resps *next;
} RESPONSES;
```

Figura 7.32: Struct responses

A continuación se explican los campos:

- **ip:** Contiene la dirección IPv4 o IPv6 de quien generó la respuesta.
- **buf:** Apuntador al buffer que contiene el paquete respuesta.
- **cBit:** Este bit indica si la respuesta tenía el bit C (ver Figura 4.1) activado.
- **id:** Contiene el ID del paquete.
- **ipLen:** Tamaño del campo *ip*.
- **bufLen:** Tamaño del buffer *buf*.
- **anCount:** Número de respuestas indicado en el campo *ANCOUNT* (ver Figura 4.1).
- **fromIface:** Contiene el identificador de la interfaz.

El campo *ip* y el campo *id* se utilizan para descartar respuestas duplicadas. El campo *cBit* es muy importante. Como se dijo anteriormente, si un Responder tiene múltiples interfaces conectadas a la misma red (interfaces “espejo”), cada vez que responda una consulta generara múltiples respuestas, una por cada interfaz espejo. Para evitar confundir al Sender, y que este genere un alerta de conflicto, el bit C es colocado en 1, para indicar que podría recibir múltiples respuestas, en cuyo caso el Sender debe tratar cada respuesta con el bit C encendido como una misma respuesta. Evidentemente, si se reciben múltiples respuestas, algunas con el bit C encendido y otras no, entonces se está ante la presencia de un conflicto, lo cual debe ser alertado por el Sender.

Una vez recibida una respuesta (o varias si fuese el caso) y descartado un conflicto y/o respuestas duplicadas, se tomará, tal como lo especifica el RFC 4795, como respuesta válida, la primera recibida.

Es importante destacar lo siguiente, una respuesta puede contener múltiples respuestas, es decir, puede contener múltiples registros (cada registro se considera una respuesta) DNS. Por ejemplo, si se realiza una consulta de nombres por el registro AAAA (IPv6) y se obtiene una respuesta, una interfaz que contenga varias IPv6 responderá con tantos registros AAAA como direcciones IPv6 tenga la interfaz, donde cada registro se considera una respuesta. En la Figura 7.33 se muestra la diferencia entre respuestas.

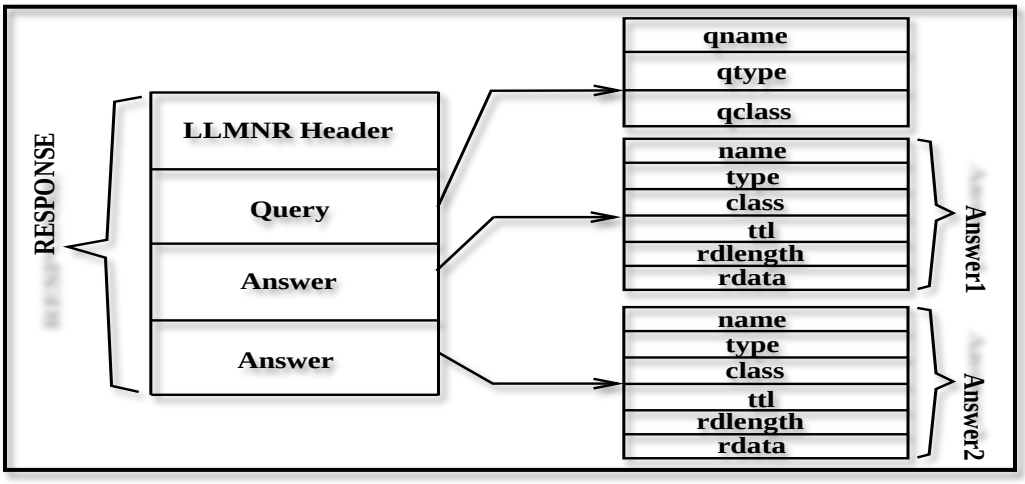


Figura 7.33: Ejemplo de Respuesta

Una vez obtenida la respuesta definitiva, esta será procesada y cada respuesta será almacenada en una lista enlazada. En la Figura 7.34 se muestra la estructura de cada respuesta.

```
typedef struct answs{
        unsigned short TYPE;
        int TTL;
        unsigned short RDLENGTH;
        void *RDATA;
        struct answs *next;
}; ANSWERS
```

Figura 7.34: Struct answers

- **Fase de Codificación:** A continuación se muestra el prototipo de la función *startS2()* (Figura 7.35).

```
int startS2(const ARGS *args, NETIFACE *ifs, ANSWERS
*answers)
```

Figura 7.35: Función startS2

Esta función se encarga de realizar la consulta (query) requerida, seleccionar una respuesta (response) y procesar y almacenar cada registro (answer). *args* es una estructura que contiene una lista de parámetros, entre estos se encuentra el nombre y el tipo que se quiere consultar, *ifs* es la lista de interfaces por donde sacar la consulta y *answers* es la lista para almacenar las respuestas (registros).

Haciendo uso de la función *socket()* se crean 2 sockets (IPv4 e IPv6) para enviar un query al nombre y tipo que se quiere. Con la función *setsockopt()* junto a las opciones *IPPROTO_IP*, *IP_MULTICAST_IF* (para IPv4) e *IPPROTO_IPV6, IPV6_MULTICAST_IF* (para IPv6) *se selecciona* por cual interfaz de red saldrá el paquete, en este caso se usará como interfaz de salida la primera de la lista *ifs.* Posterior al envío del mensaje, se invoca a la función *recvMsg()* que se encargará de recibir las posibles respuestas. La manera como funciona *recvMsg* es similar a lo explicado en la Sección 7.2.4.

Cada respuesta recibida es almacenada en una lista de tipo *RESPONSES.* La inserción y remoción de nodos se hace con las primitivas *responsesListHead(), newResponse(), deleteResponsesFrom().* Una vez se tiene todas las respuesta se verifica con la primitiva *haveConflict()* si existe un conflicto, es decir si para la consulta hecha se obtuvo 2 o más respuestas provenientes de distintos orígenes. En cuyo caso el mensaje de alerta de conflicto es generado y enviado y la consulta se considera sin respuesta. Si no existe conflicto entonces con la primitiva *haveValidResponse*() se verifica si la respuesta es negativa[2], en cuyo caso se considera sin respuesta. En caso contrario, si se recibe una respuesta válida, entonces con las primitivas *getAnswers(), newAnswer()* se obtienen y almacenan todos los registros (respuestas) en la lista *answers*, y se da por finalizada (resuelta) la consulta. Es importante destacar que en caso de que una consulta falle, es decir, no se reciba respuesta o se recibe una respuesta negativa, el proceso se repetirá en la siguiente interfaz de la lista *ifs*, hasta que se resuelva la consulta o hasta que no haya más interfaces disponibles.

- **Fase de Pruebas:** Para verificar la correctitud del proceso de generación de consultas y el procesamiento y almacenamiento de respuestas se realizó, con la ayuda de herramientas de programación tan útiles como *valgrind* y *gdb*, pruebas unitarias sobre las primitivas que se encargan del manejo de la listas de respuestas (responses) y registros (answers). Es importante recordar que parte del código de esta iteración es código reutilizado (y debidamente probado) del Responder. También se montaron diversos y múltiples escenarios de pruebas que permitieron validar que las correctas acciones fueron ejecutadas a la hora de intentar resolver una consulta sobre una o varias interfaces de red.

7.3.2 Iteración 2: Integración con el Sistema Operativo

El protocolo LLMNR es un servicio de resolución de nombres de equipo, que se ofrece al sistema operativo y a las aplicaciones, generalmente, a través de este último. En algunos sistemas operativos, agregar un nuevo servicio de resolución de nombres implicaría recompilar el kernel del mismo. Afortunadamente GNU/Linux ofrece un diseño que permite integrar un nuevo servicio de resolución de nombres con relativa facilidad.

[2] Se considera una respuesta negativa a aquella respuesta recibida con 0 registros DNS.

- **Fase de Diseño:** Para integrar un nuevo servicio de resolución de nombres, GNU/Linux usa la herramienta *NSSWITCH* (name server switch). *NSSWITCH* es un archivo de configuración, ubicado en /etc/nsswitch.conf, que usa la librería *glibc* (GNU Lib C) para determinar las fuentes de las cuales obtener la resolución de nombres. *NSSWITCH* está dividido en categorías (o bases de datos) y cada categoría tiene una lista de fuentes en orden de preferencia. En la Figura 7.36 se puede observar un ejemplo de un archivo *nsswitch.conf*

```
# /etc/nsswitch.conf
# Example configuration of GNU Name Service Switch
# functionality.

passwd:         compat
group:          compat
shadow:         compat
gshadow:        compat
hosts:          files myhostname dns llmnr
networks:       files
services:       db files
```

Figura 7.36: Ejemplo nsswitch.conf

En este caso, las categorías (o base de datos) son *passwd, group, shadow, gshadow, hosts, networks* y *services*. Las fuentes serian *compat, files, myhostname, dns* y *db.* Para el caso de resolución de nombres de equipo, la categoría es *hosts,* por lo que el primer paso para integrar el Sender consiste en agregar la cadena "llmnr" como fuente dentro de la categoría *hosts,* por supuesto este no es el único paso.

Antes de continuar hablando sobre la herramienta *NSSWITCH* es importante destacar lo siguiente, en última instancia, la resolución de nombres de equipo, o dominio, es un servicio que presta el sistema operativo a las aplicaciones, por lo que es este último el que tiene que encargarse de resolver los nombres, pero, la manera en como el servicio es prestado a las aplicaciones es a través de *glibc* y una serie de primitivas o funciones básicas que las aplicaciones deben/pueden invocar y cuyos prototipos se listan a continuación:

- struct hostent *gethostbyname_r();
- struct hostent *gethostbyname2_r();
- struct hostent *gethostbyaddr_r();

Para resolver registros *A se usa* la función *gethostbyname_r()* o *gethostbyname2_r().* Para registros *AAAA* solo sirve la función *gethostbyname2_r().* Para registros *PTR* se usa la función *gethostbyaddr_r()*. Es preciso decir que aparte de estas 3 funciones, *glibc* incorpora otra serie de funciones que realizan la misma tarea. Alguna de estas son *gethostbyname(), gethostbyadr(), getaddrinfo(),* etc, sin embargo, estas funciones son funciones "wrappers"[3] de las 3 funciones básicas mencionadas anteriormente. Para el resto de registros (*SOA; NS, MX;* etc) GNU/Linux no ofrece soporte nativo, y aunque

[3] Una función wrapper (envoltorio) es una función o subrutina que llama a otra subrutina sin la necesidad de computo adicional.

existen librerías de C que pueden usarse, estas son añadidos a *glibc* y no parte del sistema estándar.

El segundo paso para integrar un nuevo servicio de resolución de nombres es crear una librería dinámica[4]. El nombre de esta librería debe seguir la siguiente convención: *libnss_service.so.2,* donde "service" es el nombre del servicio o la fuente que se encargará de resolver el nombre. Esta librería contendrá las 3 funciones básicas mencionadas anteriormente que quieran desarrollarse para el nuevo servicio. El nombre de las funciones debe seguir la siguiente convención: _nss_service_function.

Para el caso de LLMNR (y para poner un ejemplo), el nombre de la librería será l*ibnss_llmnr_so.2*, y las funciones básicas quedan de la siguiente manera:

- _nss_llmnr_gethostbyname_r();
- _nss_llmnr_gethostbyname2_r();
- _nss_llmnr_gethostbyaddr_r();

En la Figura 7.37 se muestra el prototipo completo de las 3 funciones para el módulo *libnss_llmnr.so.2*. Es preciso aclarar que no necesariamente las 3 funciones deben estar definidas dentro de la librería dinámica, y en caso que se invoque una función sin definir o no existente, se tomará como respuesta no encontrada y se procederá a usar el siguiente servicio en la lista.

```
enum nss_status _nss_llmnr_gethostbyname_r(const char *name,
                                           struct hostent *result_buf,
                                           ……char *buf, size_t buflen,
                                           ….. int *errnop, int *h_errnop);

enum nss_status _nss_llmnr_gethostbyname_r(const char *name, int af,
                                           struct hostent *result_buf,
                                           ……char *buf, size_t buflen,
                                           ….. int *errnop, int *h_errnop);

enum nss_status _nss_llmnr_gethostbyaddr_r(const void *addr,
                                              socklen_t len, int format,
                                              struct hostent *result_buf,
                                              char *buffer, size_t buflen,
                                              int *errnop, int *h_errnop);
```

Figura 7.37: libnss_llmnr.so2 prototipos

En la Figura 7.38 se muestra un ejemplo que intenta ilustrar cómo funciona la resolución de nombres de equipo o dominio bajo GNU/Linux y cómo se puede incorporar un servicio para dicha tarea. En el ejemplo, una aplicación intenta resolver el registro *A* para el nombre *name.* En el sistema existen 3 servicios o fuentes para la resolución de nombres de equipo, *service1, service2* (ambos fallan en la resolución) y LLMNR.

[4] Los detalles de qué es y cómo se crea una librería dinámica son excluidos. Todos los detalles al respecto puede encontrarse en "The Linux Documentation Project" (http://tldp.org)

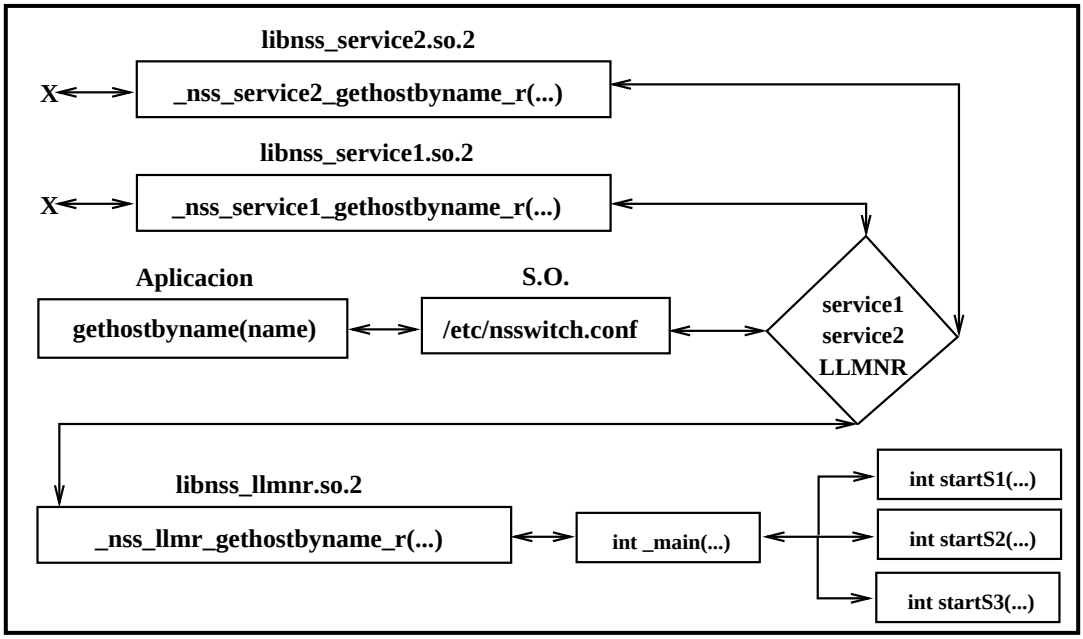


Figura 7.38: Resolución de Nombres bajo GNU/Linux

- **Fase de Codificación:** Una vez que es solicitado el servicio de resolución de nombres de equipo o dominio, la respuesta, de conseguirse alguna, debe retornarse en una estructura *hostent*. Esta estructura es definida por la biblioteca de sockets de C, y es común a las 3 funciones. En la Figura 7.39 se puede apreciar la estructura *hostent.*

```
struct hostent {
        char *h_name;
        char **h_aliases;
        int h_addrtype;
        int h_length;
        char **h_addr_list;
}
```

Figura 7.39: Struct hostent

Una vez el sistema operativo invoca el servicio LLMNR, el primer paso es construir una estructura *args*. Esta estructura es de tipo personalizada y se usa para agrupar bajo un mismo nombre la larga lista de parámetros que recibe cualquiera de las 3 funciones básicas (ver Figura 7.37). Luego se invoca a la función *_main()*, que tal como su nombre lo indica es un pseudo main (las librerías dinámicas no tienen función *main*). A grandes rasgos esta función realiza 2 pasos:

- Invocar la función que generará el proceso de consulta LLMNR (descrito en la Sección 7.2.2)
- Invocar la función que se encargará de llenar la estructura *hostent* y retornar la respuesta.

La función encargada de construir o llenar la estructura *hostent* contentiva de la respuesta final es la función *startS3().* En la Figura 7.40 se puede apreciar el prototipo de la función.

```
int startS3(const ARGS *args, ANSWERS *answers);
```

Figura 7.40: Función startS3

Donde *args* es la lista de parámetros de la función básica y *answers* es la lista enlazada que contiene los registros o respuestas conseguidos. Como se dijo anteriormente, la respuesta se retorna en una estructura *hostent,* sin embargo, *hostent* es solo una estructura que contiene apuntadores y no contiene un buffer en donde almacenar las respuestas. El buffer, o espacio de memoria donde se almacena, es provisto por el usuario y viene indicado por el parámetro *char *buf* y el tamaño del mismo viene indicado por el parámetro *size_t buflen* (ver Figura 7.37). En caso de que el tamaño del buffer sea menor al necesario para almacenar las respuestas se retorna el valor *NSS_STATUS_TRYAGAIN* y se le asigna a *herrnop* (este es pasado por referencia) el valor *ERANGE.*

Para almacenar la(s) respuestas(s) en el buffer no existe una convención específica en cuanto a cómo deben ir organizadas las mismas. En la Figura 7.41 se muestra el esquema escogido para almacenar las respuestas a consultas *A* y *AAAA*. Igualmente, también se muestra el esquema escogido para almacenar las respuestas para consultas *PTR.* Es preciso recordar que una consulta de tipo *A* o *AAAA* puede generar varias respuestas, es decir, para un nombre pueden existir varias direcciones IP o una relación 1:N, con N >= 1. Análogamente para consultas *PTR,* pero la relación es varios nombres o seudónimos para una IP.

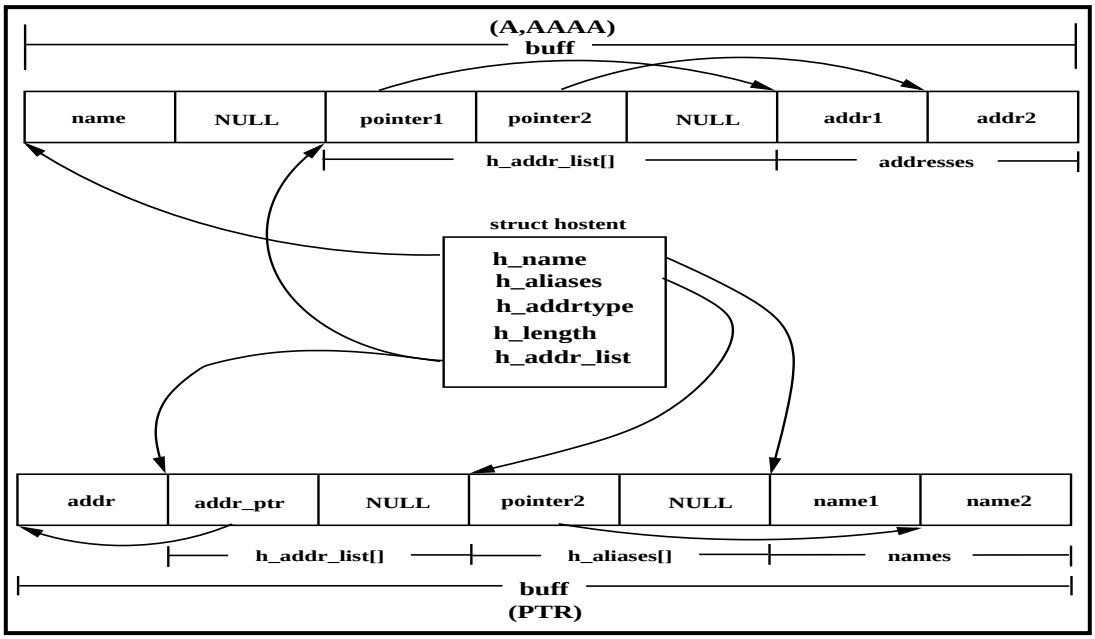


Figura 7.41: Esquema Hostent (A, AAAA, PTR)

- **Fase de Pruebas:** Para verificar la correctitud del almacenamiento de respuestas dentro de una estructura *hostent* se realizó, con la ayuda de herramientas de programación tan útiles como *valgrind* y *gdb*, pruebas unitarias sobre las funciones encargadas de realizar el proceso. Particularmente la herramienta *valgrind* es de vital importancia a la hora de manejar apuntadores y

verificar referencias a zonas de memoria inválidas o fuera de rango. También se utilizó las opciones “Wall” y “Wextra” del compilador GCC para detectar cualquier conversión de punteros inválida o incorrecta. Igualmente se montaron escenarios de pruebas que permitieron validar la correcta integración del Sender como servicio de resolución de nombres, sobre el sistema operativo GNU/Linux.

7.3.3 Iteración 3: Desarrollo de la Cache LLMNR

Tal como especifica el RFC 4795 [**3**], el protocolo LLMNR implementa una cache a la hora de resolver nombres. Esta cache es única y separada de la cache DNS. El protocolo no es más específico al respecto, si por ejemplo, debe ser un archivo o el nombre del mismo, por lo que el diseño e implementación de la misma queda a libre albedrio.

- **Fase de Diseño:** La cache LLMNR será implementada en un archivo binario, ubicado en */tmp/llmnr.cache.* A la hora de guardar un registro o respuesta, debe existir un diseño o cabecera que permita recuperar posteriormente la información guardada. En la Figura 7.42 se muestra el diseño escogido para almacenar los registros dentro del archivo cache.

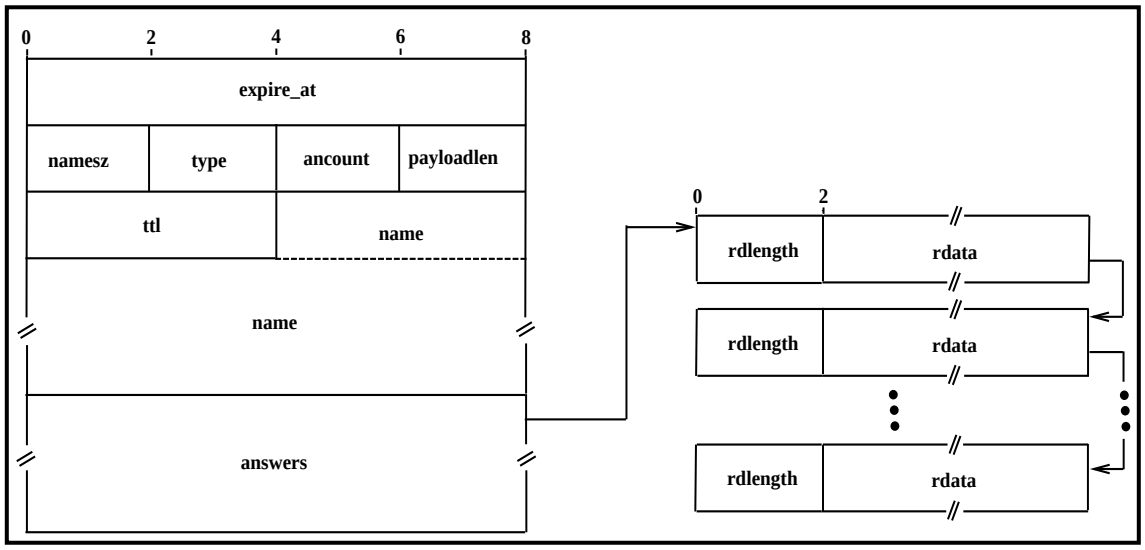


Figura 7.42: Cabecera Cache

A continuación se listan cada uno de los campos de la cabecera:

- **expire_at:** El tiempo (Unix) en la cual expira el registro.
- **namesz:** Tamaño del nombre
- **type:** Tipo de registro (*A, AAA*, etc).
- **ancount;** Número de respuestas que guarda el registro.
- **payloadlen:** Tamaño del payload. El payload comienza en las respuestas.
- **ttl:** Tiempo de vida (time to live) del registro.
- **name:** El nombre de equipo consultado.
- **answers:** Los registros consultados. Es preciso destacar que no se guarda el registro completo (ver Figura 7.6), solo la parte de la data correspondiente a la respuesta (*rdlength* y *rdata).*

- **Fase de Codificación:** Para poder acceder al archivo que contiene la cache, primero es necesario bloquear el archivo, esto debido a que el Sender es invocado a través de la librería dinámica de nombre *libnss_llmnr.so.2*, lo cual significa que

pueden haber múltiples instancias del Sender corriendo. Para bloquear y desbloquear el archivo de la cache se usa las funciones *oPenAndGetLock()* y *releaseFileLock().* La llamada *oPenAndGetLock()* es de tipo no bloqueante, es decir, si falla en obtener el "lock" del archivo entonces retorna inmediatamente, permitiendo que el Sender continúe su proceso. Para agregar o buscar un registro se usan las funciones *cacheAnswers()* y *getCachedAnswer()* respectivamente.

El Sender, antes de lanzar una consulta LLMNR, primero buscará la respuesta en la cache y para evitar prolongar el tiempo de búsqueda, es necesario mantener actualizada la cache, es decir, cada cierto tiempo borrar registros expirados. Para determinar cuándo es necesario actualizar la cache se usa el macro *CLEANCACHE*, el cual define el número máximo de registros que puede contener la cache (estos se cuentan cuando se está realizando una búsqueda), antes de proceder a actualizarla. Para realizar dicha actualización se invoca la función *cleanCache()*. Esta se encargará de leer uno a uno los registros en la cache e ir eliminando aquellos registros expirados, con la ayuda del campo *expire_at* de la cabecera de la cache (ver Figura 7.42). Similarmente, esta función debe bloquear el archivo de la cache antes de poder editarlo, para ello usa la función *getLockNb()*. Esta llamada es de tipo no bloqueante, lo que significa que si la función falla en obtener el "lock" del archivo de la cache, se aborta el proceso de limpieza (el cual deberá reintentarse en la próxima invocación del Sender), permitiendo que el Sender finalice y retorne una respuesta (es preciso destacar que la invocación a la función *cleanCache* es el ultimo paso realizado por el Sender).

- **Fase de Pruebas:** Para verificar el correcto funcionamiento de la implementación y manejo de la cache LLMNR, se realizaron pruebas unitarias sobre las funciones encargadas de llevar a cabo las operaciones sobre la misma. Se montaron escenarios de prueba que permitieron evaluar y validar el correcto acceso y manejo de la cache así como la correcta lectura y escritura del archivo cache. Estos escenarios tomaron en cuenta ambientes de alta concurrencia.

8. **Pruebas de Validación y Análisis de Resultados**

Con el objetivo de validar el correcto desempeño del demonio LLMNR, se plantean diversos escenarios de pruebas para evaluar, de manera integral, los distintos aspectos presentes en el protocolo LLMNR, tanto en el Sender como en el Responder. Estos escenarios de pruebas se realizaron en un laboratorio de Comunicación y Redes (LACORE) ubicado en la Facultad de Ciencias de la Universidad Central de Venezuela. A continuación se presentan los escenarios junto con una breve descripción de los objetivos de los mismos y los resultados obtenidos.

8.1 Pruebas de Funcionamiento:

El primer conjunto de pruebas (escenario 1, escenario 2, escenario 3 y escenario 4) están enfocadas en validar el funcionamiento y/o funcionalidad básica del demonio desarrollado (*llmnrd*).

8.1.1 Escenario 1

En este escenario se presenta una red LAN sencilla compuesta por 3 computadores conectados a un switch Catalyst 2960 marca Cisco. Los computadores *lacore03* y *lacore04* corren el sistema operativo Debian 7.0 con el demonio *llmnrd* instalado. El computador *lacore05* corre el sistema operativo Windows 7, con el demonio *ms-llmnr* instalado. Todos los computadores tienen sola una interfaz de red conectada. En la Figura 8.1 puede verse una representación visual del escenario 1.

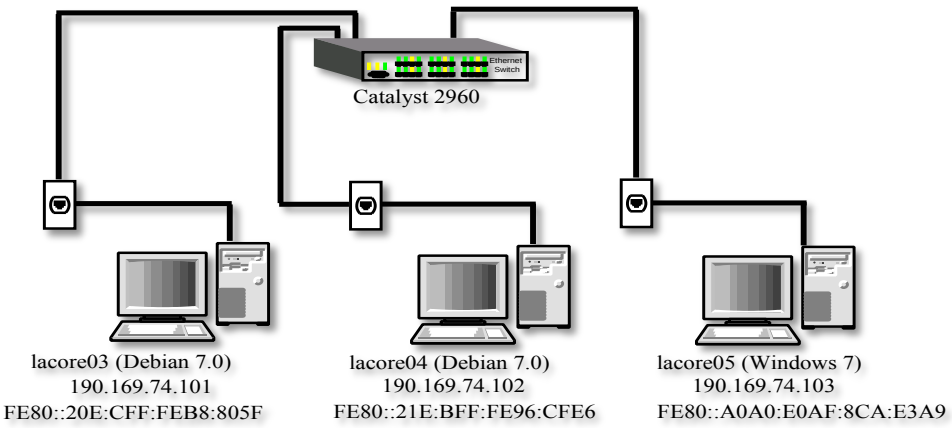


Figura 8.1: Escenario 1

A continuación se listan las pruebas realizadas en el escenario 1:

- Arranque del demonio (Responder) en *lacore03*.
- Arranque del demonio (Responder) en *lacore04*.
- Verificación de unicidad de nombre en *lacore03*.
- Verificación de unicidad de nombre en *lacore04*.
- Respuestas a consultas (A, AAAA) al nombre *lacore03*.
- Respuestas a consultas (A, AAAA) al nombre *lacore04*.
- Agregación de direcciones IPv6 en *lacore03*.
- Agregación de direcciones IPv6 en *lacore04*.

Con la ayuda del script *llmnr,* el comando *ps* y *grep,* puede arrancarse el demonio *llmnrd* (Responder) y verificar su estatus. El script *llmnr,* ubicado en el directorio */etc/init.d* puede llevar a cabo las siguientes acciones:

- **start:** Arranca el demonio *llmnrd* (Responder).
- **stop:** Detiene el demonio *llmnrd* (Responder).
- **reload:** Reinicia el demonio *llmnrd* (Responder).
- **status:** Verifica si el demonio *llmnrd* está corriendo (Responder).

En la Figura 8.2 y la Figura 8.3 se puede apreciar el estatus y el proceso de arranque del demonio *llmnrd,* en la máquina *lacore03* y *lacore04*, respectivamente. Es preciso destacar que también se hizo uso del comando *ps* en conjunción con el comando *grep,* el cual sirve para revalidar que, de hecho, no hay ningún proceso de nombre "llmnrd" corriendo dentro del sistema.

Figura 8.2: Arranque del Demonio en lacore03 (Escenario 1)

Figura 8.3: Arranque del Demonio en lacore04 (Escenario 1)

En la Figura 8.4 y la Figura 8.5 se muestra el proceso de verificación de unicidad de nombre, una vez iniciado el demonio, en *lacore03* y *lacore04* respectivamente.

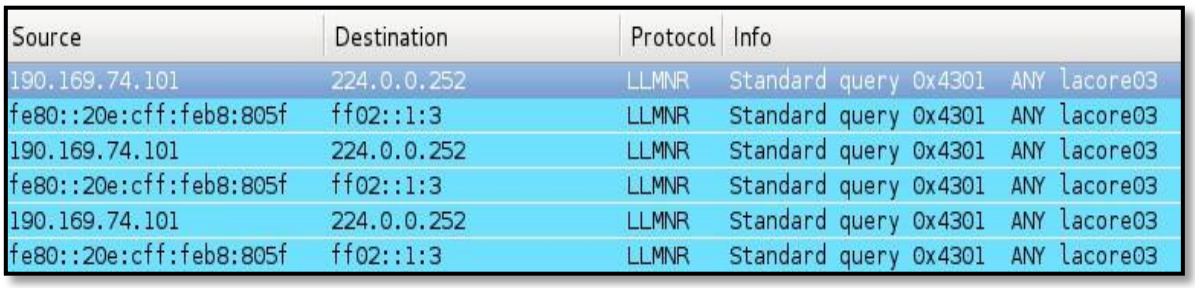
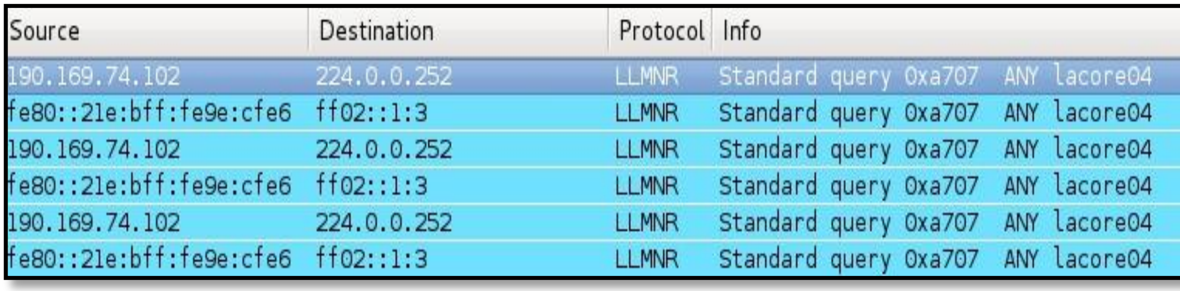


Figura 8.4: Verificación de Unicidad de Nombre en lacore03 (Escenario 1)



Source	Destination	Protocol	Info
190.169.74.102	224.0.0.252	LLMNR	Standard query 0xa707 ANY lacore04
fe80::21e:bff:fe9e:cfe6	ff02::1:3	LLMNR	Standard query 0xa707 ANY lacore04
190.169.74.102	224.0.0.252	LLMNR	Standard query 0xa707 ANY lacore04
fe80::21e:bff:fe9e:cfe6	ff02::1:3	LLMNR	Standard query 0xa707 ANY lacore04
190.169.74.102	224.0.0.252	LLMNR	Standard query 0xa707 ANY lacore04
fe80::21e:bff:fe9e:cfe6	ff02::1:3	LLMNR	Standard query 0xa707 ANY lacore04

Figura 8.5: Verificación de Unicidad de Nombre en lacore04 (Escenario 1)

En la Figura 8.8 y la Figura 8.9 se muestra el resultado de respuestas a consultas A y AAAA al nombre *lacore03* y *lacore04*. Las consultas se hicieron con el comando *ping* (mostrado en la Figura 8.6 y Figura 8.7) desde la máquina *lacore05* [5].

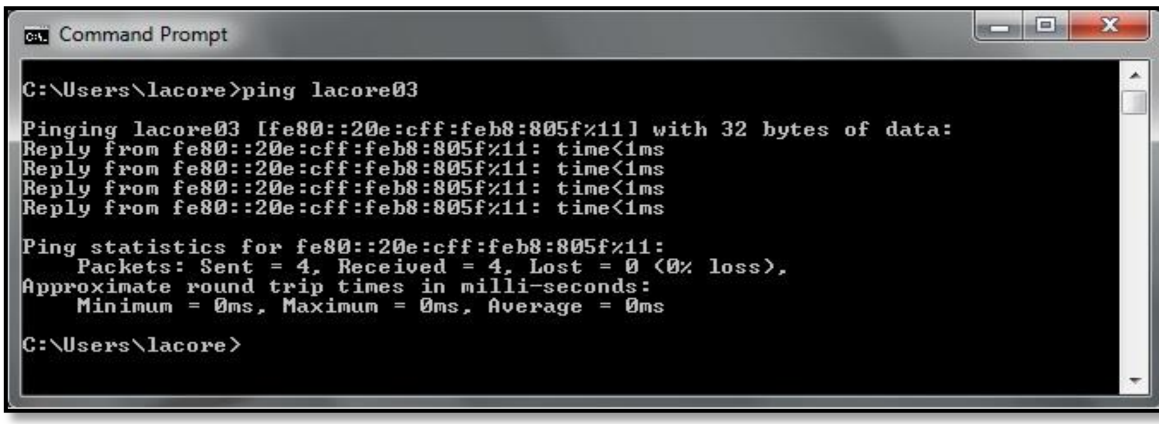


Figura 8.6: Ping 1 a lacore03 (Escenario 1)

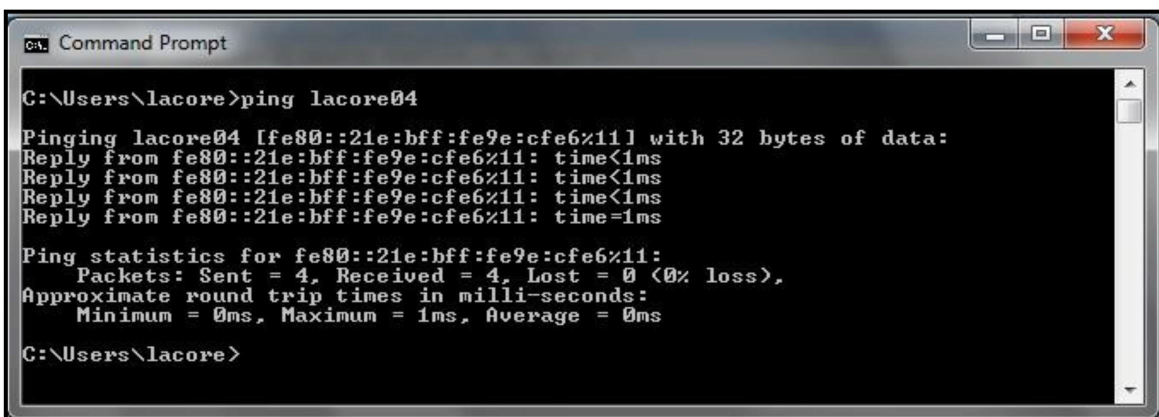
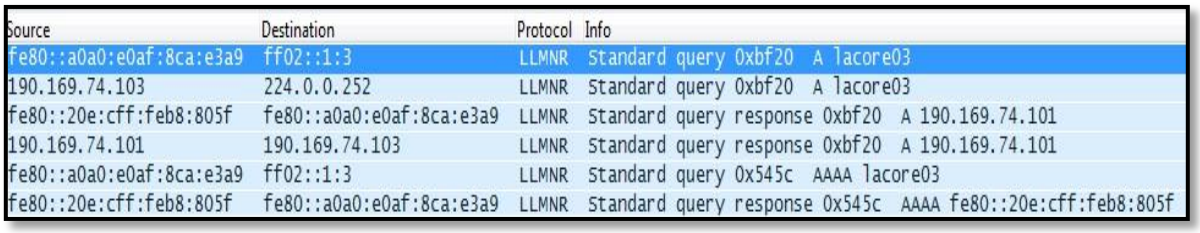


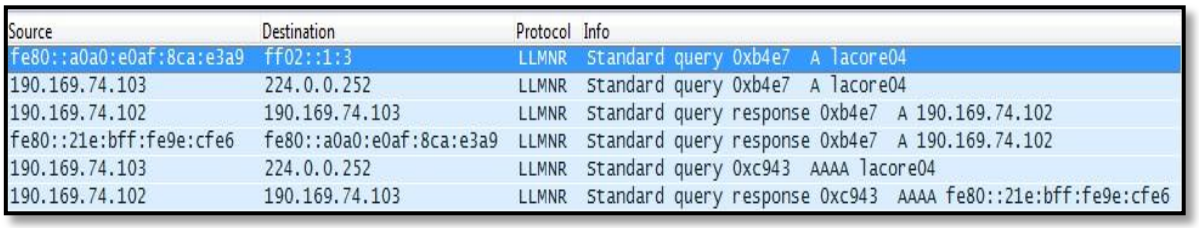
Figura 8.7: Ping 1 a lacore04 (Escenario 1)

[5] Cabe aclarar que *lacore05* recibe respuestas *A* y *AAAA* pero se favorece IPv6 (respuestas *AAAA*) tal como está especificado en el RFC 6724.



Source	Destination	Protocol	Info
fe80::a0a0:e0af:8ca:e3a9	ff02::1:3	LLMNR	Standard query 0xbf20 A lacore03
190.169.74.103	224.0.0.252	LLMNR	Standard query 0xbf20 A lacore03
fe80::20e:cff:feb8:805f	fe80::a0a0:e0af:8ca:e3a9	LLMNR	Standard query response 0xbf20 A 190.169.74.101
190.169.74.101	190.169.74.103	LLMNR	Standard query response 0xbf20 A 190.169.74.101
fe80::a0a0:e0af:8ca:e3a9	ff02::1:3	LLMNR	Standard query 0x545c AAAA lacore03
fe80::20e:cff:feb8:805f	fe80::a0a0:e0af:8ca:e3a9	LLMNR	Standard query response 0x545c AAAA fe80::20e:cff:feb8:805f

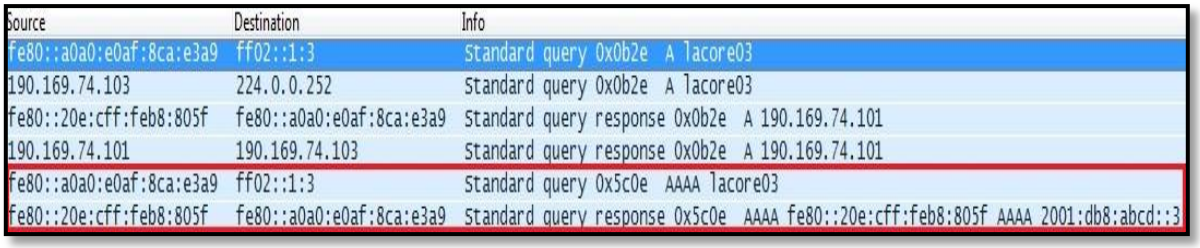
Figura 8.8: Respuesta 1 de lacore03 (Escenario 1)



Source	Destination	Protocol	Info
fe80::a0a0:e0af:8ca:e3a9	ff02::1:3	LLMNR	Standard query 0xb4e7 A lacore04
190.169.74.103	224.0.0.252	LLMNR	Standard query 0xb4e7 A lacore04
190.169.74.102	190.169.74.103	LLMNR	Standard query response 0xb4e7 A 190.169.74.102
fe80::21e:bff:fe9e:cfe6	fe80::a0a0:e0af:8ca:e3a9	LLMNR	Standard query response 0xb4e7 A 190.169.74.102
190.169.74.103	224.0.0.252	LLMNR	Standard query 0xc943 AAAA lacore04
190.169.74.102	190.169.74.103	LLMNR	Standard query response 0xc943 AAAA fe80::21e:bff:fe9e:cfe6

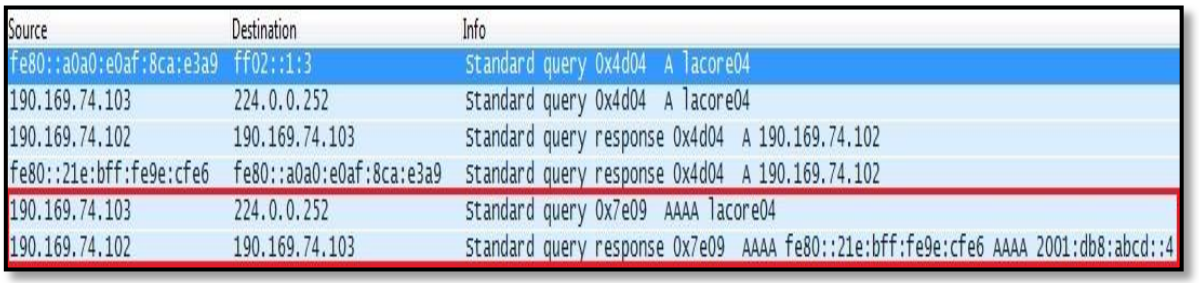
Figura 8.9: Respuesta 1 de lacore04 (Escenario 1)

Se agregó la dirección IPv6 2001:DB8:ABCD::3/64 y 2001:DB8:ABCD::4/64 a la máquina *lacore03* y *lacore04* respectivamente. Se realizó, nuevamente, un *ping* desde la máquina *lacore05* a ambas máquinas. En la Figura 8.10 y la Figura 8.11 se muestran los resultados obtenidos. Aquí se puede observar que para las consultas *AAAA, lacore03* y *lacore04* contestan con las 2 direcciones IPv6.



Source	Destination	Info
fe80::a0a0:e0af:8ca:e3a9	ff02::1:3	Standard query 0x0b2e A lacore03
190.169.74.103	224.0.0.252	Standard query 0x0b2e A lacore03
fe80::20e:cff:feb8:805f	fe80::a0a0:e0af:8ca:e3a9	Standard query response 0x0b2e A 190.169.74.101
190.169.74.101	190.169.74.103	Standard query response 0x0b2e A 190.169.74.101
fe80::a0a0:e0af:8ca:e3a9	ff02::1:3	Standard query 0x5c0e AAAA lacore03
fe80::20e:cff:feb8:805f	fe80::a0a0:e0af:8ca:e3a9	Standard query response 0x5c0e AAAA fe80::20e:cff:feb8:805f AAAA 2001:db8:abcd::3

Figura 8.10: Respuesta 2 de lacore03 (Escenario 1)



Source	Destination	Info
fe80::a0a0:e0af:8ca:e3a9	ff02::1:3	Standard query 0x4d04 A lacore04
190.169.74.103	224.0.0.252	Standard query 0x4d04 A lacore04
190.169.74.102	190.169.74.103	Standard query response 0x4d04 A 190.169.74.102
fe80::21e:bff:fe9e:cfe6	fe80::a0a0:e0af:8ca:e3a9	Standard query response 0x4d04 A 190.169.74.102
190.169.74.103	224.0.0.252	Standard query 0x7e09 AAAA lacore04
190.169.74.102	190.169.74.103	Standard query response 0x7e09 AAAA fe80::21e:bff:fe9e:cfe6 AAAA 2001:db8:abcd::4

Figura 8.11: Respuesta 2 de lacore04 (Escenario 1)

8.1.2 Escenario 2

En este escenario se presenta una red LAN sencilla compuesta por 3 computadores, *lacorexyz* (izquierda), *lacore04* y *lacorexyz* (derecha). Todos conectados a un router inalámbrico WRT54G marca Linksys. La máquina *lacorexyz* (izquierda) y *lacore04* corren el sistema operativo Debian 7.0 y el demonio *llmnrd* instalado. La máquina *lacorexyz* (derecha) corre el sistema operativo Windows 7 con el demonio *ms-llmnr* instalado. Todos

los computadores tienen una sola interfaz de red (cableada) conectada. En la Figura 8.12 puede verse una representación visual del escenario 2.

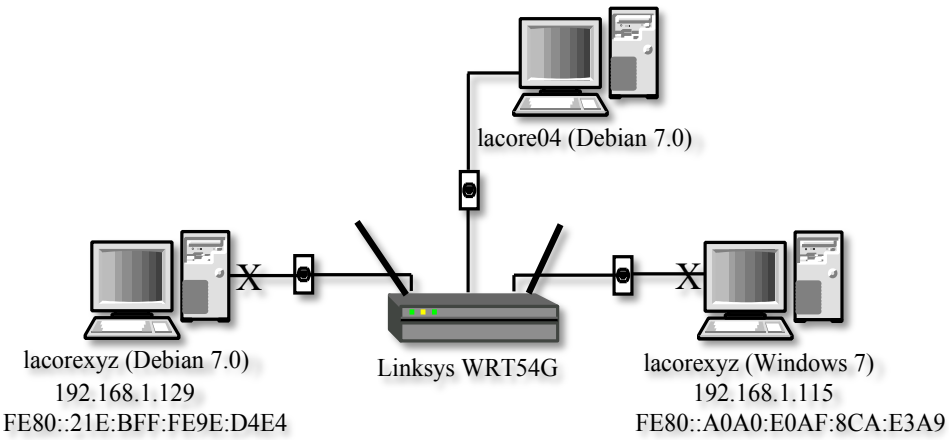


Figura 8.12: Escenario 2

A continuación se listan las pruebas realizadas en el escenario 2:

- Conexión simultanea de *lacorexyz* (izquierda y derecha) al router inalámbrico WRT54G marca Linksys.
- Defensa y resolución de conflicto del nombre *lacorexyz*.
- Consultas (A, AAA) al nombre *lacorexyz*.

En la Figura 8.13 y la Figura 8.14 se muestra la configuración de las interfaces de red de *lacorexyz* (izquierda y derecha respectivamente).

```
lacore@lacorexyz:~$ /sbin/ifconfig
eth1      Link encap:Ethernet  HWaddr 00:1e:0b:9e:d4:e4
          inet addr:192.168.1.129  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::21e:bff:fe9e:d4e4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7428 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2421 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1094236 (1.0 MiB)  TX bytes:951328 (929.0 KiB)
          Interrupt:17

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:550 errors:0 dropped:0 overruns:0 frame:0
          TX packets:550 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:42772 (41.7 KiB)  TX bytes:42772 (41.7 KiB)

lacore@lacorexyz:~$
```

Figura 8.13: Interfaces de lacorexyz izquierda (Escenario 2)

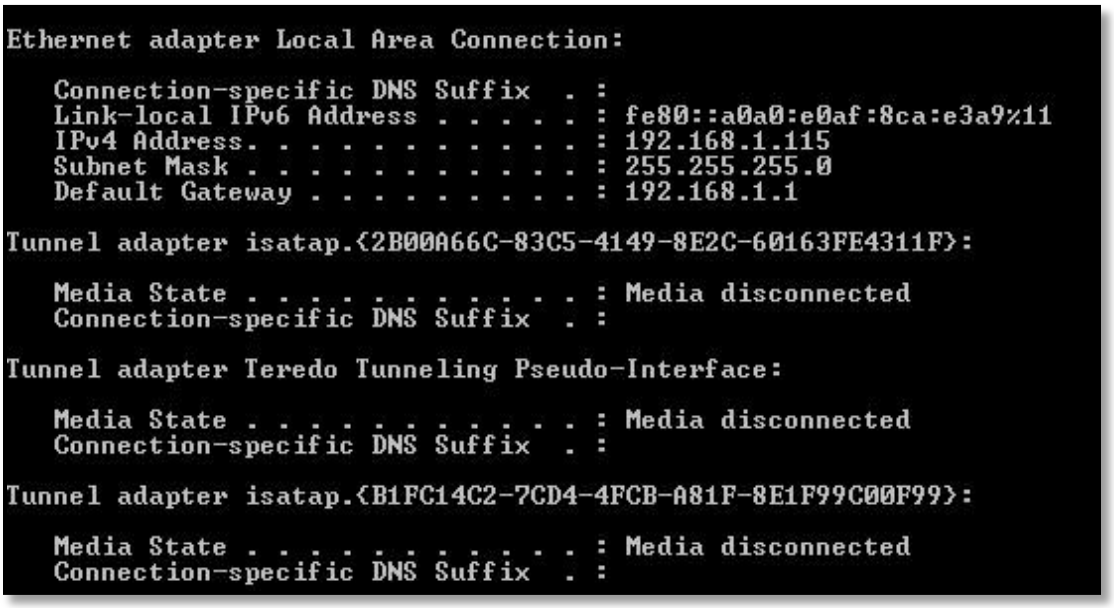


Figura 8.14: Interfaces de lacorexyz derecha (Escenario 2)

En la Figura 8.15 se muestra la captura de pantalla del tráfico LLMNR que se generó al conectar ambas máquinas *lacorexyz*. Tal como puede apreciarse, el demonio *llmnrd* (lacorexyz izquierda) es el primero en levantar y responder. El demonio *ms-llmnr* no alcanza a responder, puesto que recibe una respuesta (tentativa) proveniente de una máquina con una IPv6 lexicográficamente menor.

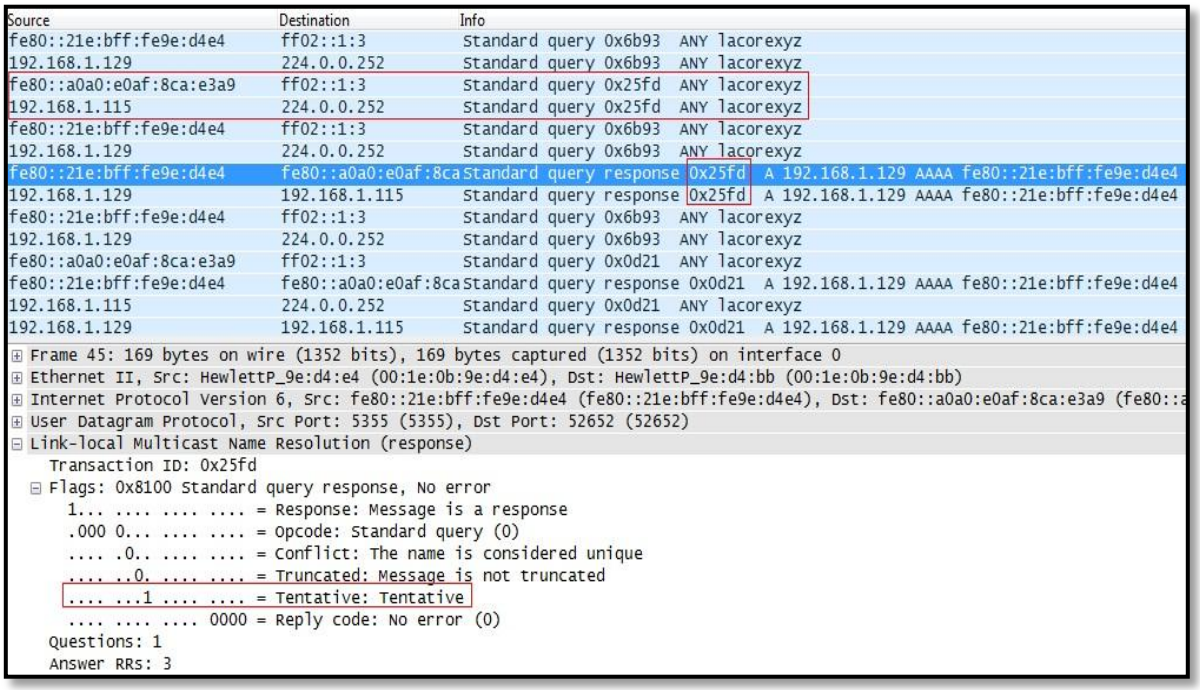


Figura 8.15: Verificación 1 de Unicidad de Nombre lacorexyz (Escenario 2)

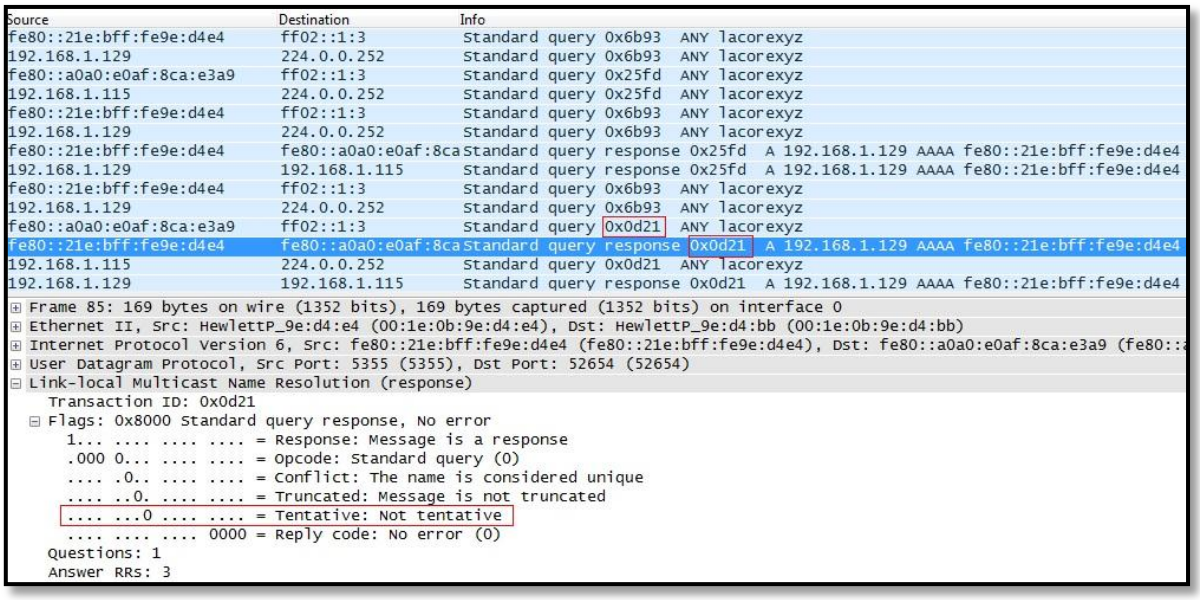


Figura 8.16: Verificación 2 de Unicidad de Nombre lacorexyz (Escenario 2)

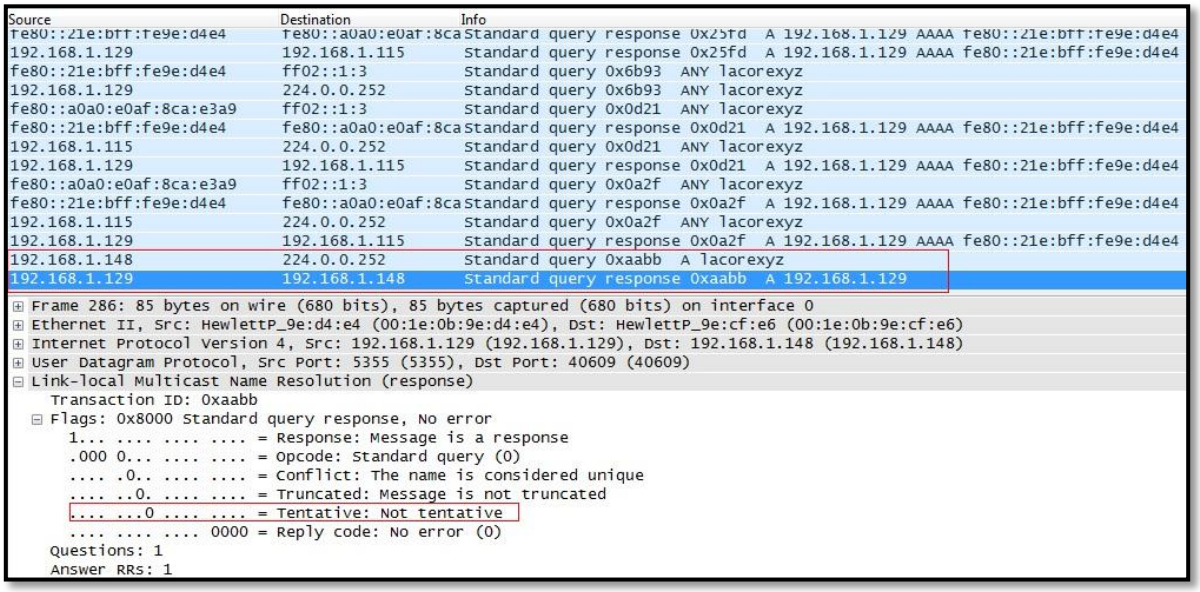


Figura 8.17: LLMNR Query lacorexyz (Escenario 2)

En la Figura 8.16 y la Figura 8.17 puede apreciarse que efectivamente la máquina *lacorexyz* (izquierda) ganó el conflicto y pasa a ser autoritativa del nombre.

8.1.3 Escenario 3

Este escenario está compuesto por 2 redes LAN (LAN1 y LAN2) y 3 computadores, *lacore03*, *lacore04* y *lacore05*. *lacore03* y *lacore05* están conectados a un switch Catalyst 2960 marca Cisco, la primera a través de 2 interfaces de red, y la segunda a través de una única interfaz de red. *lacore03* y *lacore04* están conectados a un router inalámbrico WRT54G marca Linksys a través de una interfaz de red inalámbrica. *lacore03* y *lacore04* corren el sistema operativo Debian 7.0 con el demonio *llmnrd* instalado. *lacore05* corre el

sistema operativo Windows 7 con el demonio *ms-llmnr* instalado. En la Figura 8.18, puede verse una representación visual del escenario 3.

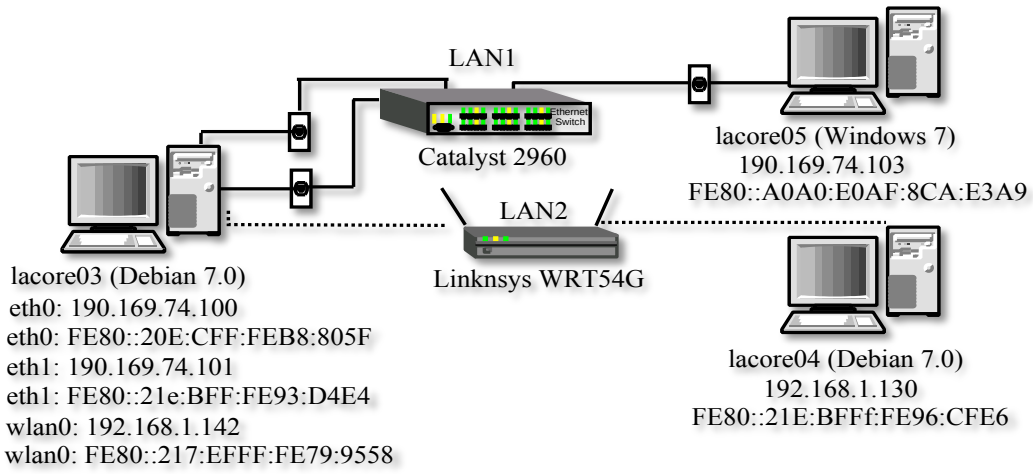


Figura 8.18: Escenario 3

A continuación se listan las pruebas realizadas en el escenario 3:

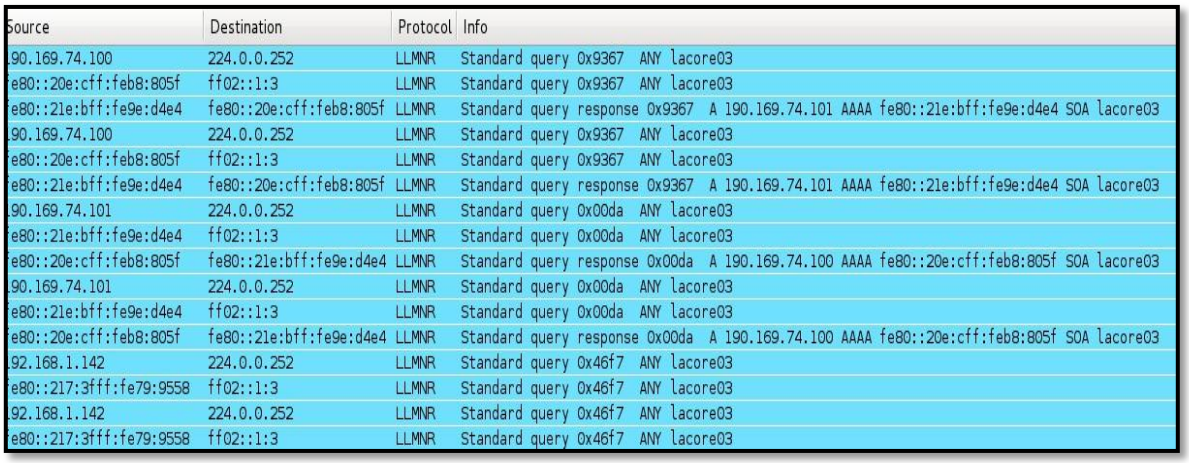
- Arranque y verificación de unicidad de nombre en *lacore03*.
- Respuestas a consultas (A, AAAA) en *lacore03*.

En la Figura 8.19 se muestra la configuración de interfaces de *lacore03*

```
root@lacore03:/home/lacore# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0e:0c:b8:80:5f
          inet addr:190.169.74.100  Bcast:190.169.74.255  Mask:255.255.255.0
          inet6 addr: fe80::20e:cff:feb8:805f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:107810 errors:0 dropped:0 overruns:0 frame:0
          TX packets:69206 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:76671323 (73.1 MiB)  TX bytes:8933191 (8.5 MiB)

eth1      Link encap:Ethernet  HWaddr 00:1e:0b:9e:d4:e4
          inet addr:190.169.74.101  Bcast:190.169.74.255  Mask:255.255.255.0
          inet6 addr: fe80::21e:bff:fe9e:d4e4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:64228 errors:0 dropped:0 overruns:0 frame:0
          TX packets:35631 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:61616878 (58.7 MiB)  TX bytes:4356252 (4.1 MiB)
          Interrupt:17

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:164 errors:0 dropped:0 overruns:0 frame:0
          TX packets:164 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:10032 (9.7 KiB)  TX bytes:10032 (9.7 KiB)

wlan0     Link encap:Ethernet  HWaddr 00:17:3f:79:95:58
          inet addr:192.168.1.142  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::217:3fff:fe79:9558/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:654 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1638 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:191819 (187.3 KiB)  TX bytes:188413 (183.9 KiB)
```

Figura 8.19: Interfaces de lacore03 (escenario 3)

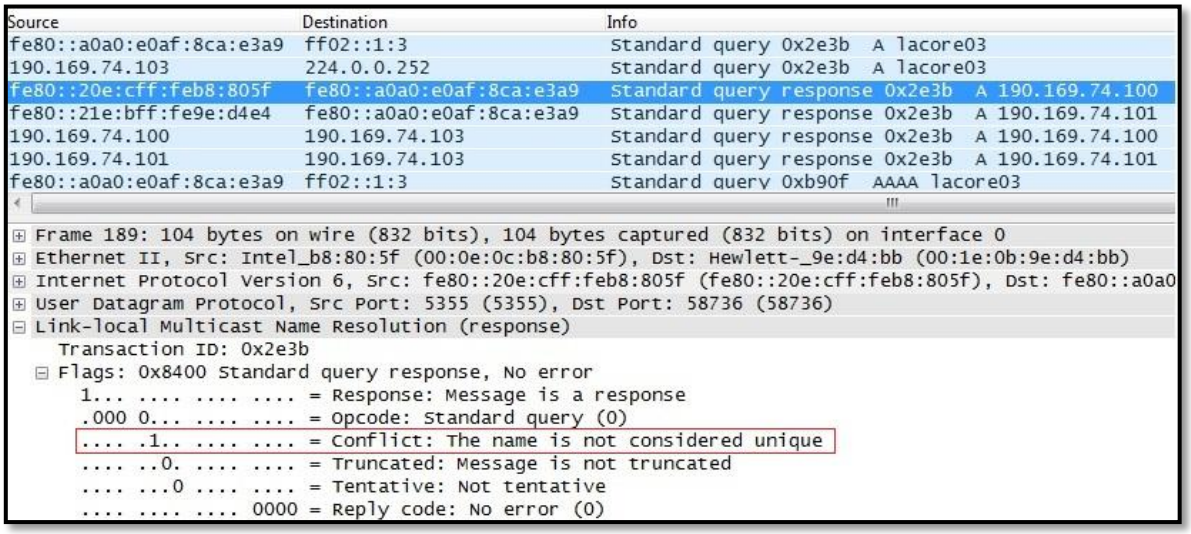
En la Figura 8.20 se muestra la captura de pantalla del tráfico LLMNR que se generó al momento de arrancar el demonio *llmnrd* en *lacore03*. Es preciso destacar que para este escenario se redujo el valor de *QUERYMAXTRIES* (macro que define el número de intentos) de 3 a 2.



Source	Destination	Protocol	Info
190.169.74.100	224.0.0.252	LLMNR	Standard query 0x9367 ANY lacore03
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x9367 ANY lacore03
fe80::21e:bff:fe9e:d4e4	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x9367 A 190.169.74.101 AAAA fe80::21e:bff:fe9e:d4e4 SOA lacore03
190.169.74.100	224.0.0.252	LLMNR	Standard query 0x9367 ANY lacore03
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x9367 ANY lacore03
fe80::21e:bff:fe9e:d4e4	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x9367 A 190.169.74.101 AAAA fe80::21e:bff:fe9e:d4e4 SOA lacore03
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x00da ANY lacore03
fe80::21e:bff:fe9e:d4e4	ff02::1:3	LLMNR	Standard query 0x00da ANY lacore03
fe80::20e:cff:feb8:805f	fe80::21e:bff:fe9e:d4e4	LLMNR	Standard query response 0x00da A 190.169.74.100 AAAA fe80::20e:cff:feb8:805f SOA lacore03
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x00da ANY lacore03
fe80::21e:bff:fe9e:d4e4	ff02::1:3	LLMNR	Standard query 0x00da ANY lacore03
fe80::20e:cff:feb8:805f	fe80::21e:bff:fe9e:d4e4	LLMNR	Standard query response 0x00da A 190.169.74.100 AAAA fe80::20e:cff:feb8:805f SOA lacore03
192.168.1.142	224.0.0.252	LLMNR	Standard query 0x46f7 ANY lacore03
fe80::217:3fff:fe79:9558	ff02::1:3	LLMNR	Standard query 0x46f7 ANY lacore03
192.168.1.142	224.0.0.252	LLMNR	Standard query 0x46f7 ANY lacore03
fe80::217:3fff:fe79:9558	ff02::1:3	LLMNR	Standard query 0x46f7 ANY lacore03

Figura 8.20: Verificación de Unicidad de Nombre en lacore03 (Escenario 3)

En la Figura 8.21 se muestra un *ping* realizado a *lacore03* y la correspondiente captura del tráfico de red a través de *wireshark* (todo desde *lacore05*).



Source	Destination	Info
fe80::a0a0:e0af:8ca:e3a9	ff02::1:3	Standard query 0x2e3b A lacore03
190.169.74.103	224.0.0.252	Standard query 0x2e3b A lacore03
fe80::20e:cff:feb8:805f	fe80::a0a0:e0af:8ca:e3a9	Standard query response 0x2e3b A 190.169.74.100
fe80::21e:bff:fe9e:d4e4	fe80::a0a0:e0af:8ca:e3a9	Standard query response 0x2e3b A 190.169.74.101
190.169.74.100	190.169.74.103	Standard query response 0x2e3b A 190.169.74.100
190.169.74.101	190.169.74.103	Standard query response 0x2e3b A 190.169.74.101
fe80::a0a0:e0af:8ca:e3a9	ff02::1:3	Standard query 0xb90f AAAA lacore03

```
⊞ Frame 189: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0
⊞ Ethernet II, Src: Intel_b8:80:5f (00:0e:0c:b8:80:5f), Dst: Hewlett-_9e:d4:bb (00:1e:0b:9e:d4:bb)
⊞ Internet Protocol Version 6, Src: fe80::20e:cff:feb8:805f (fe80::20e:cff:feb8:805f), Dst: fe80::a0a0
⊞ User Datagram Protocol, Src Port: 5355 (5355), Dst Port: 58736 (58736)
⊟ Link-local Multicast Name Resolution (response)
    Transaction ID: 0x2e3b
  ⊟ Flags: 0x8400 Standard query response, No error
      1... .... .... .... = Response: Message is a response
      .000 0... .... .... = Opcode: Standard query (0)
      .... .1.. .... .... = Conflict: The name is not considered unique
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...0 .... .... = Tentative: Not tentative
      .... .... .... 0000 = Reply code: No error (0)
```

Figura 8.21: Respuesta 1 de lacore03 (Escenario 3)

Es importante resaltar que en la cabecera en la respuesta, el bit *conflict* está marcado en la respuesta. *lacore03* está conectada a la red LAN1 a través de 2 interfaces de red (interfaces espejo). Esto generara 2 respuestas (provenientes de cada una de las interfaces de red conectadas) por cada consulta, por lo que el bit *conflict* debe marcarse para alertar a quien generó la consulta, *lacore05* en este caso, que puede recibir múltiples respuestas, y que todas deben tratarse como una única respuesta, y no generar una falsa alerta de conflicto.

8.1.4 Escenario 4

En este escenario se presenta una red LAN sencilla, compuesta por 3 máquinas, todas conectadas a un switch Catalyst 2960 marca Cisco. *lacore03* y *lacore04* corren el sistema operativo Debian 7.0 con el demonio *lmnrd* instalado. *lacore05* corre el sistema operativo Windows 7, con el demonio *ms-llmnr* instalado. *lacore04* tiene un demonio apache y SSH

corriendo. Todos los computadores tienen una única interfaz de red. En la Figura 8.22 se puede apreciar una representación visual del escenario 4.

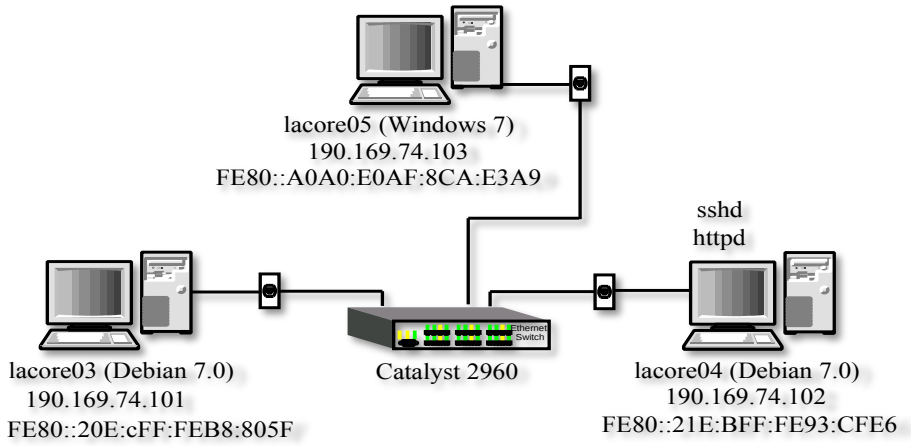


Figura 8.22: Escenario 4

A continuación se listan las pruebas realizadas en el escenario 4:

- Comando *ping* a *lacore04*, desde *lacore03*.
- Comando *ping* a *lacore05* desde *lacore03*.
- Comando *ssh* a *lacore04*, desde *lacore03*.
- Consulta HTTP (vía Mozilla Firefox) a *lacore04*, desde *lacore03*.

En la Figura 8.23 y la Figura 8.24 se muestra la configuración de interfaces de red de *lacore03* y *lacore04*, respectivamente.

```
lacore@lacore03:~$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0e:0c:b8:80:5f
          inet addr:190.169.74.101  Bcast:190.169.74.255  Mask:255.255.255.0
          inet6 addr: fe80::20e:cff:feb8:805f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:30634 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3190 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5969206 (5.6 MiB)  TX bytes:366864 (358.2 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:9600 (9.3 KiB)  TX bytes:9600 (9.3 KiB)
```

Figura 8.23: Interfaces de lacore03 (Escenario 4)

```
lacore@lacore04:~$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1e:0b:9e:cf:e6
          inet addr:190.169.74.102  Bcast:190.169.74.255  Mask:255.255.255.0
          inet6 addr: fe80::21e:bff:fe9e:cfe6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:170506 errors:0 dropped:0 overruns:0 frame:0
          TX packets:138933 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:151087816 (144.0 MiB)  TX bytes:16524198 (15.7 MiB)
          Interrupt:17

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:254 errors:0 dropped:0 overruns:0 frame:0
          TX packets:254 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:17907 (17.4 KiB)  TX bytes:17907 (17.4 KiB)

lacore@lacore04:~$
```

Figura 8.24: Interfaces de lacore04 (Escenario 4)

Haciendo uso del comando *ping* y el comando *ssh* se puede probar la integración del Sender dentro del sistema operativo. El resultado de ambos comandos, así como la correspondiente captura de tráfico (LLMNR, ICMP, SSH) que generó, se muestra en la Figura 8.25, la Figura 8.26, la Figura 8.27, la Figura 8.28, la Figura 8.29 y la Figura 8.30.

```
lacore@lacore03:~$ ping -c 4 lacore04
PING lacore04 (190.169.74.102) 56(84) bytes of data.
64 bytes from lacore04 (190.169.74.102): icmp_req=1 ttl=64 time=0.136 ms
64 bytes from lacore04 (190.169.74.102): icmp_req=2 ttl=64 time=0.145 ms
64 bytes from lacore04 (190.169.74.102): icmp_req=3 ttl=64 time=0.142 ms
64 bytes from lacore04 (190.169.74.102): icmp_req=4 ttl=64 time=0.145 ms

--- lacore04 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.136/0.142/0.145/0.003 ms
lacore@lacore03:~$
```

Figura 8.25: Ping a lacore04 (Escenario 4)

Source	Destination	Protocol	Info
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x4567 A lacore04
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x4567 A lacore04
190.169.74.102	190.169.74.101	LLMNR	Standard query response 0x4567 A 190.169.74.102
fe80::21e:bff:fe9e:cfe6	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x4567 A 190.169.74.102
190.169.74.101	190.169.74.102	ICMP	Echo (ping) request id=0x4748, seq=1/256, ttl=64
190.169.74.102	190.169.74.101	ICMP	Echo (ping) reply id=0x4748, seq=1/256, ttl=64
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x23c6 PTR 102.74.169.190.in-addr.arpa
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x23c6 PTR 102.74.169.190.in-addr.arpa
190.169.74.102	190.169.74.101	LLMNR	Standard query response 0x23c6 PTR lacore04
fe80::21e:bff:fe9e:cfe6	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x23c6 PTR lacore04
190.169.74.101	190.169.74.102	ICMP	Echo (ping) request id=0x4748, seq=2/512, ttl=64
190.169.74.102	190.169.74.101	ICMP	Echo (ping) reply id=0x4748, seq=2/512, ttl=64
190.169.74.101	190.169.74.102	ICMP	Echo (ping) request id=0x4748, seq=3/768, ttl=64
190.169.74.102	190.169.74.101	ICMP	Echo (ping) reply id=0x4748, seq=3/768, ttl=64
190.169.74.101	190.169.74.102	ICMP	Echo (ping) request id=0x4748, seq=4/1024, ttl=64
190.169.74.102	190.169.74.101	ICMP	Echo (ping) reply id=0x4748, seq=4/1024, ttl=64

Figura 8.26: Respuesta ping lacore04 (Escenario 4)

```
lacore@lacore03:~$ ping -c 4 lacore05
PING lacore05 (190.169.74.103) 56(84) bytes of data.
64 bytes from lacore05 (190.169.74.103): icmp_req=1 ttl=128 time=0.158 ms
64 bytes from lacore05 (190.169.74.103): icmp_req=2 ttl=128 time=0.170 ms
64 bytes from lacore05 (190.169.74.103): icmp_req=3 ttl=128 time=0.158 ms
64 bytes from lacore05 (190.169.74.103): icmp_req=4 ttl=128 time=0.142 ms

--- lacore05 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.142/0.157/0.170/0.010 ms
lacore@lacore03:~$
```

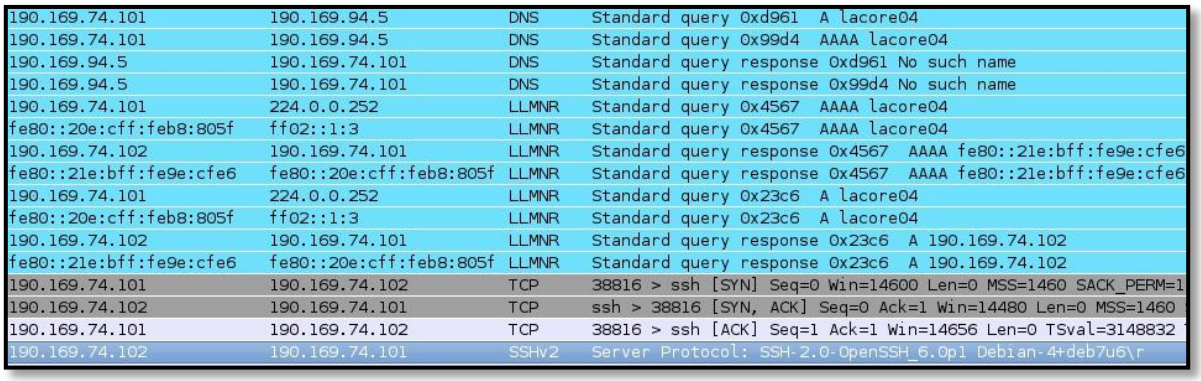
Figura 8.27: Ping a lacore05 (Escenario 4)

Source	Destination	Protocol	Info
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x4567 A lacore05
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x4567 A lacore05
190.169.74.103	190.169.74.101	LLMNR	Standard query response 0x4567 A 190.169.74.103
fe80::a0a0:e0af:8ca:e3a9	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x4567 A 190.169.74.103
190.169.74.101	190.169.74.103	ICMP	Echo (ping) request id=0x55ce, seq=1/256, ttl=64
190.169.74.103	190.169.74.101	ICMP	Echo (ping) reply id=0x55ce, seq=1/256, ttl=128
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x23c6 PTR 103.74.169.190.in-addr.arpa
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x23c6 PTR 103.74.169.190.in-addr.arpa
190.169.74.103	190.169.74.101	LLMNR	Standard query response 0x23c6 PTR lacore05
fe80::a0a0:e0af:8ca:e3a9	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x23c6 PTR lacore05
190.169.74.101	190.169.74.103	ICMP	Echo (ping) request id=0x55ce, seq=2/512, ttl=64
190.169.74.103	190.169.74.101	ICMP	Echo (ping) reply id=0x55ce, seq=2/512, ttl=128
190.169.74.101	190.169.74.103	ICMP	Echo (ping) request id=0x55ce, seq=3/768, ttl=64
190.169.74.103	190.169.74.101	ICMP	Echo (ping) reply id=0x55ce, seq=3/768, ttl=128
190.169.74.101	190.169.74.103	ICMP	Echo (ping) request id=0x55ce, seq=4/1024, ttl=64
190.169.74.103	190.169.74.101	ICMP	Echo (ping) reply id=0x55ce, seq=4/1024, ttl=128

Figura 8.28: Respuesta ping lacore05 (Escenario 4)

```
lacore@lacore03:~$ ssh lacore@lacore04
The authenticity of host 'lacore04 (190.169.74.102)' can't be established.
ECDSA key fingerprint is 68:78:41:fb:92:04:13:26:ab:91:8a:d2:a5:ad:24:09.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'lacore04,190.169.74.102' (ECDSA) to the list of known hosts.
lacore@lacore04's password:
Linux lacore04 3.2.0-4-amd64 #1 SMP Debian 3.2.78-1 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 31 15:31:39 2016 from 190.169.74.101
lacore@lacore04:~$ exit
logout
Connection to lacore04 closed.
lacore@lacore03:~$
```

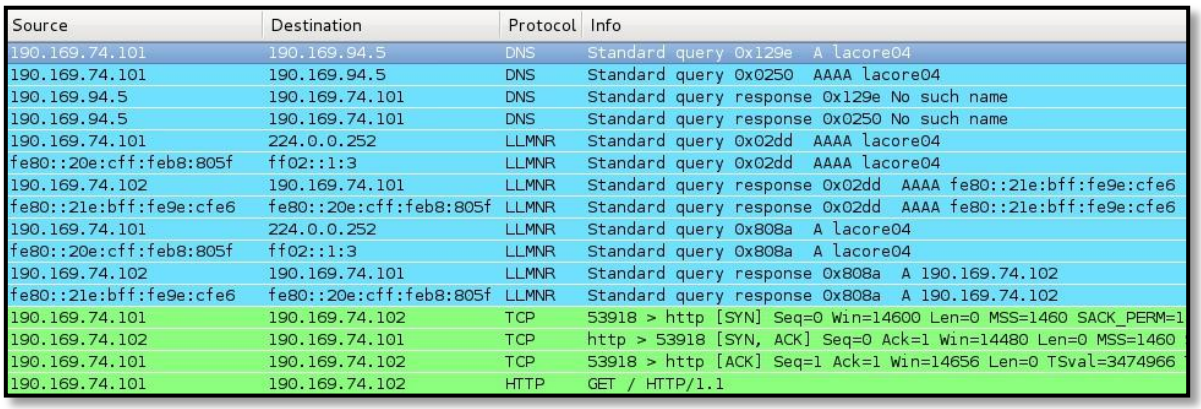
Figura 8.29: SSH a lacore04 (Escenario 4)



190.169.74.101	190.169.94.5	DNS	Standard query 0xd961 A lacore04
190.169.74.101	190.169.94.5	DNS	Standard query 0x99d4 AAAA lacore04
190.169.94.5	190.169.74.101	DNS	Standard query response 0xd961 No such name
190.169.94.5	190.169.74.101	DNS	Standard query response 0x99d4 No such name
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x4567 AAAA lacore04
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x4567 AAAA lacore04
190.169.74.102	190.169.74.101	LLMNR	Standard query response 0x4567 AAAA fe80::21e:bff:fe9e:cfe6
fe80::21e:bff:fe9e:cfe6	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x4567 AAAA fe80::21e:bff:fe9e:cfe6
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x23c6 A lacore04
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x23c6 A lacore04
190.169.74.102	190.169.74.101	LLMNR	Standard query response 0x23c6 A 190.169.74.102
fe80::21e:bff:fe9e:cfe6	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x23c6 A 190.169.74.102
190.169.74.101	190.169.74.102	TCP	38816 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1
190.169.74.102	190.169.74.101	TCP	ssh > 38816 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
190.169.74.101	190.169.74.102	TCP	38816 > ssh [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=3148832
190.169.74.102	190.169.74.101	SSHv2	Server Protocol: SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u6\r

Figura 8.30: Tráfico Correspondiente a SSH (Escenario 4)

Finalmente, el navegador Mozilla Firefox, que a pesar de integrar su propio cliente DNS (casi todos los grandes navegadores integran clientes DNS), cuando este falla, también hace uso del sistema de resolución de nombres que brinda el sistema operativo. En la Figura 8.31 puede apreciarse el tráfico que se generó al intentar acceder a lacore04 (este corre un demonio Apache) desde el navegador.



Source	Destination	Protocol	Info
190.169.74.101	190.169.94.5	DNS	Standard query 0x129e A lacore04
190.169.74.101	190.169.94.5	DNS	Standard query 0x0250 AAAA lacore04
190.169.94.5	190.169.74.101	DNS	Standard query response 0x129e No such name
190.169.94.5	190.169.74.101	DNS	Standard query response 0x0250 No such name
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x02dd AAAA lacore04
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x02dd AAAA lacore04
190.169.74.102	190.169.74.101	LLMNR	Standard query response 0x02dd AAAA fe80::21e:bff:fe9e:cfe6
fe80::21e:bff:fe9e:cfe6	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x02dd AAAA fe80::21e:bff:fe9e:cfe6
190.169.74.101	224.0.0.252	LLMNR	Standard query 0x808a A lacore04
fe80::20e:cff:feb8:805f	ff02::1:3	LLMNR	Standard query 0x808a A lacore04
190.169.74.102	190.169.74.101	LLMNR	Standard query response 0x808a A 190.169.74.102
fe80::21e:bff:fe9e:cfe6	fe80::20e:cff:feb8:805f	LLMNR	Standard query response 0x808a A 190.169.74.102
190.169.74.101	190.169.74.102	TCP	53918 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1
190.169.74.102	190.169.74.101	TCP	http > 53918 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
190.169.74.101	190.169.74.102	TCP	53918 > http [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=3474966
190.169.74.101	190.169.74.102	HTTP	GET / HTTP/1.1

Figura 8.31: Tráfico Correspondiente a HTTP lacore04 (Escenario 4)

8.2 Herramienta de Medición

Antes de continuar hablando de las pruebas restantes (rendimiento y estrés), es importante hablar primero de la herramienta de medición. Tal como se mencionó en la Sección 2.3.2, está entre los objetivos específicos desarrollar una herramienta cuya función consiste en proveer los mecanismos necesarios para medir rendimiento y estrés sobre *llmnrd* y *ms-llmnr* (con el objetivo de probarlos y compararlos).

Son varias las características presentes en la herramienta de medición, algunas son globales y otras son particulares de una prueba en específico. Las características mencionadas a continuación son características globales para todas las pruebas de rendimiento y estrés.

- Toda prueba está integrada por 2 o más máquinas, donde cada máquina tomará uno, y solo uno de los siguientes roles:
  - **Principal**: La máquina que corre la herramienta de medición y que se encarga de ejecutar y controlar la prueba. Solo una máquina toma este rol.
  - **Objetivo(s)**: La máquina que corre un demonio LLMNR a ser probado (*llmnrd* y *ms-llmnr*). Una o más máquinas pueden tomar este rol.

- Se usa el archivo de nombre *names1,* ubicado en el mismo directorio donde se encuentra el ejecutable de la herramienta de medición (*mtool*), para especificar el nombre de la(s) máquina(s) objetivo.
- Existe una serie de opciones de línea de comandos que sirven para especificar lo siguiente:
  - El tipo de prueba que se quiere realizar.
  - Parámetros de configuración inherentes a una prueba en específico.
- Las opciones de línea de comandos que especifican el tipo de prueba a realizar son las siguientes:
  - **h**: Muestra un mensaje de ayuda
  - **r**: Prueba de rendimiento (tiempo de respuesta método 1).
  - **R**: Prueba de rendimiento (tiempo de respuesta método 2).
  - **c**: Número de peticiones atendidas en un tiempo 't'.
  - **d**: Prueba de estrés 1.
  - **e**: Prueba de estrés 2.
  - **f**: Prueba de estrés 3.
- La máquina *principal* queda establecida sobre la máquina que ejecute la herramienta de medición especificando alguno de los parámetros mencionados previamente.[6]

Como se mencionó previamente, existe una serie de parámetros de configuración adicionales los cuales serán mencionados y/o explicados a medida que se aborde la prueba que hace uso de ellos. Adicionalmente, al inicio de la descripción de cada prueba se explicará, de manera superficial, la metodología implementada en la herramienta de medición, para llevar a cabo dicha prueba.

8.3 Pruebas de Rendimiento

En esta sección se presentan los resultados obtenidos en los diferentes escenarios de pruebas de rendimiento. Las pruebas de rendimiento empleadas son las siguientes:

- Tiempo de respuesta (usando el método 1).
- Tiempo de respuesta (usando el método 2).
- Número de peticiones atendidas por segundo.

8.3.1 Escenario 5: Tiempo de Respuesta

Este escenario busca medir el tiempo de respuesta para *n* consultas LLMNR enviadas. Para medir el tiempo de respuesta se utiliza los métodos descritos a continuación:

- **Método 1:** Se toma una muestra de tiempo ($t1_i$) previo al envío de una consulta *i.* Se toma una segunda muestra de tiempo ($t2_i$) al obtener la correspondiente respuesta. El tiempo de respuesta individual (tr_i) viene dado por la formula $tr_i = t2_i - t1_i$.El tiempo de respuesta general (TR) viene dado por la siguiente fórmula:

$$TR = \sum_{i=0}^{n} tr_i * \frac{1}{n} = \frac{tr_1 + tr_2 + tr_3 + \cdots + tr_n}{n}$$

[6] Para todas las pruebas de rendimiento y estrés realizadas se escogió a la máquina de nombre *lacore06* como máquina *principal*.

- **Método 2:** La idea del Método 2 es minimizar el error de medición del tiempo en un intercambio consulta/respuesta. Con este fin, se realizarán varios intercambios consulta/respuesta (n) encadenados entre sí, y se divide el tiempo total del experimento por n, llegando así a un promedio del tiempo del intercambio consulta/respuesta. En otras palabras, se toma una primera muestra de tiempo (T1) al inicio del experimento. Se envía la primera consulta LLMNR. Apenás llega la respuesta de la primera consulta, se encadena con la segunda consulta. Apenás llega la segunda respuesta, se encadena con la tercera consulta. Y así sucesivamente hasta llegar a la última respuesta. Con dicha última respuesta, se toma una segunda muestra de tiempo (T2). La diferencia de las 2 muestras de tiempo se divide por n, para así obtener el tiempo promedio de un intercambio consulta/respuesta. El tiempo de respuesta general (TR) viene dado por la siguiente fórmula:

$$TR = \frac{T2 - T1}{n}$$

Cada petición o consulta se genera siguiendo las siguientes directrices:

- Se selecciona, de manera aleatoria, el valor del campo *ID* (ver Figura 4.1)
- El envío es secuencial, es decir, la petición *i* se envía luego de recibir la respuesta de la petición *i -1.*
- Una respuesta se espera hasta un tiempo máximo *LLMNR_TIMEOUT* (100 ms sugeridos en RFC 4795 [**3**]). Una vez transcurrido ese tiempo se considera que no hubo respuesta (y no se cuenta la consulta fallida como enviada).

Para esta prueba está disponible en la herramienta de medición el siguiente conjunto de parámetros adicionales de configuración:

- **p**: Número de consultas a enviar.
- **x**: Número de corridas a ejecutar, es decir, cuantas veces se va a ejecutar la prueba sobre la máquina objetivo.

Comando(s) de la prueba: **mtool –r –p 10000 –x 50** / **mtool –R –p 10000 –x 50**
Máquina *objetivo*: ***lacore03, lacore04, lacore05.***

Es importante aclarar que la razón de usar 3 máquinas *objetivos* (para esta prueba y para la prueba posterior) en lugar de una (como normalmente se esperaría) radica en la intención de agregar confiabilidad en la prueba. Es decir, puesto que ambos demonios (*llmnrd* y *ms-llmnr*) corren en sistemas operativos distintos (GNU/Linux Debian 7 y Microsoft Windows 7 respectivamente), podría darse el caso de una instalación corrupta en alguno de los 2 sistemas operativos. Esa situación (de darse) puede repercutir de manera directa (o indirecta), sobre el desempeño del demonio respectivo, siendo esto injusto y/o inaceptable.

En la Figura 8.32 se muestra el resultado de medición del tiempo de respuesta para el método 1 sobre la máquina *lacore03*.

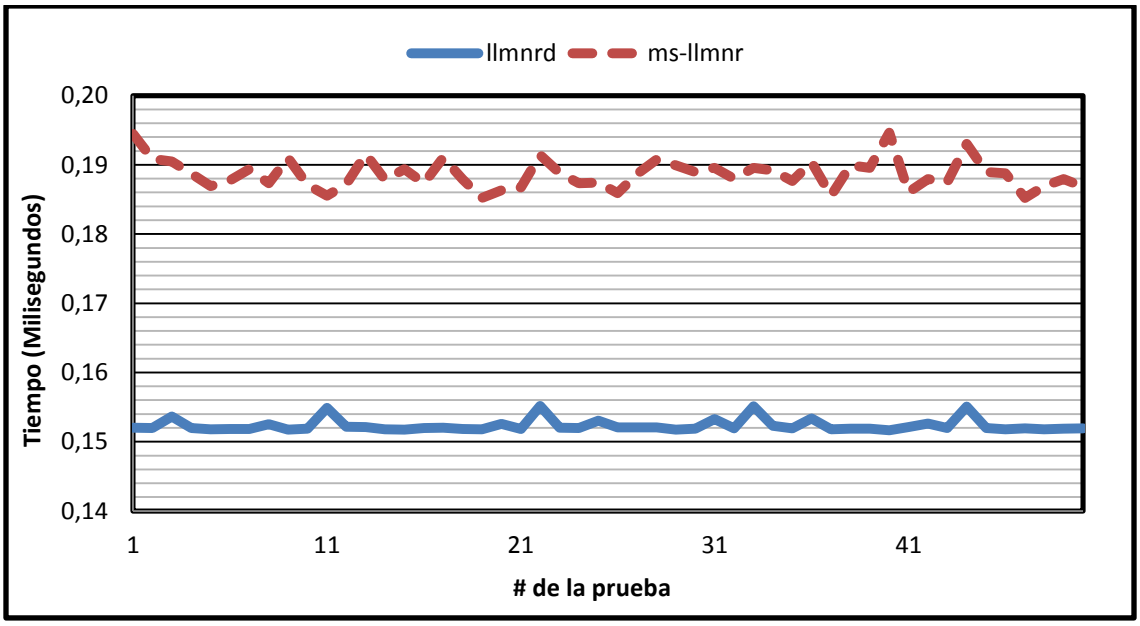


Figura 8.32: Tiempo de Respuesta Promedio (Método 1) en lacore03

Como se puede apreciar en el gráfico asociado (Figura 8.32), para esta primera medición, el demonio desarrollado (*llmnrd*) tiene un tiempo de respuesta menor al tiempo de respuesta del demonio implementado por Microsoft (*ms-llmnr*) en su sistema operativo Windows 7.

En la Figura 8.33 se muestra el resultado del tiempo de respuesta, sobre la máquina *lacore03* para ambos demonios (*llmnrd y ms-llmnr*), pero esta vez la medición se hace usando el método 2.

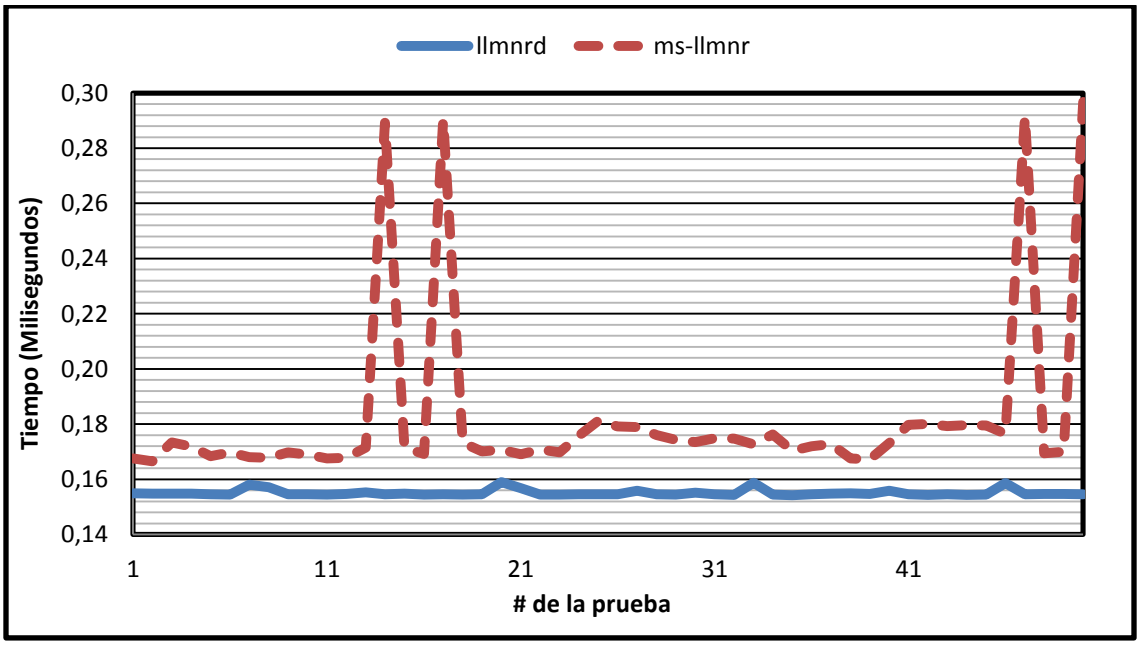


Figura 8.33: Tiempo de Respuesta Promedio (Método 2) en lacore03

En la Figura 8.34, la Figura 8.35, la Figura 8.36 y la Figura 8.37 se muestra la misma medición (método 1 y método 2) sobre las máquinas *lacore04* y *lacore05, respectivamente.*

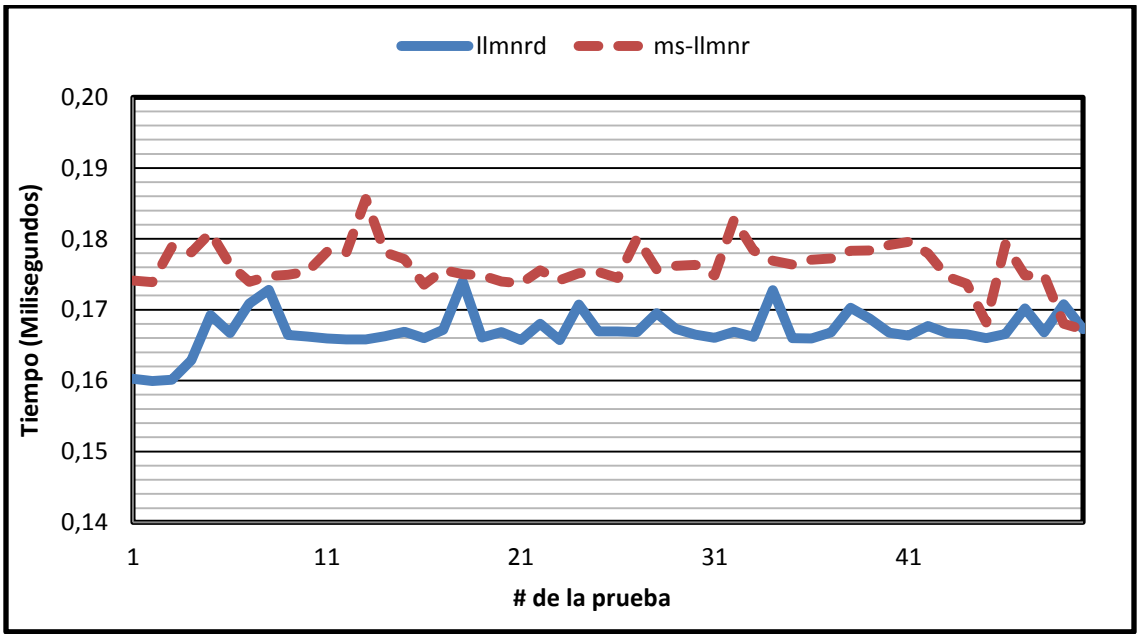


Figura 8.34: Tiempo de Respuesta Promedio (Método 1) en lacore04

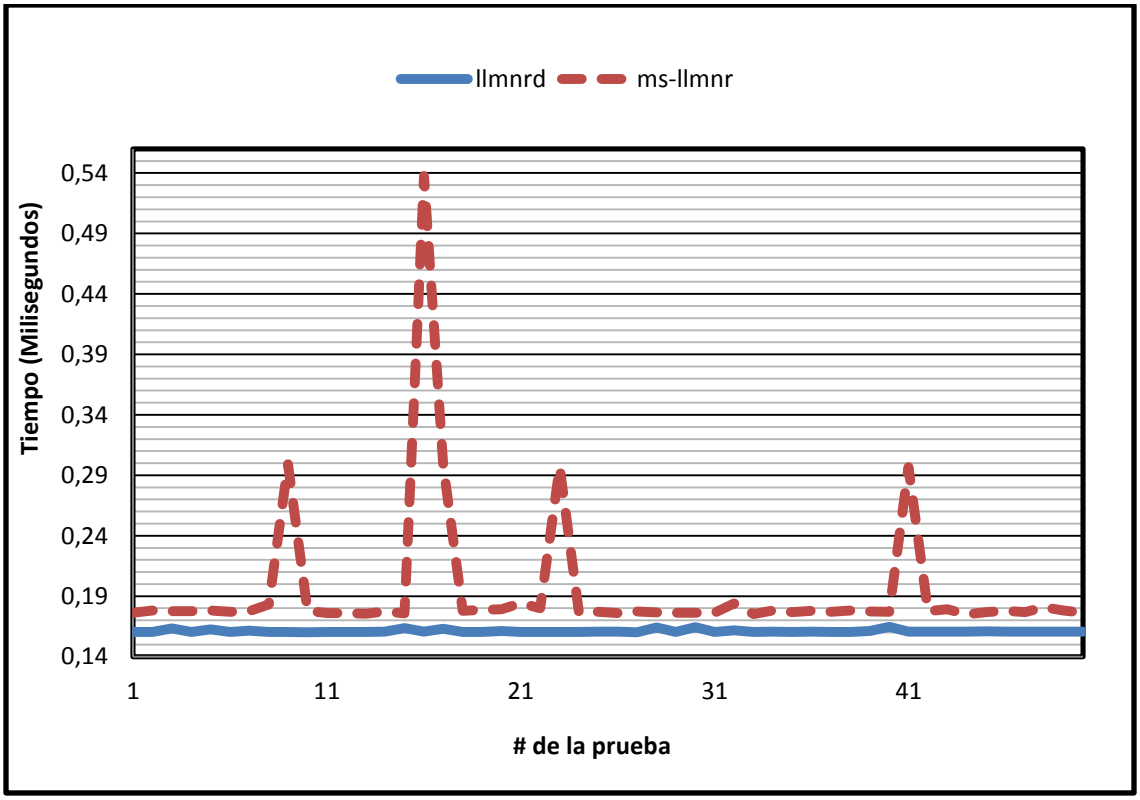


Figura 8.35: Tiempo de Respuesta Promedio (Método 2) en lacore04

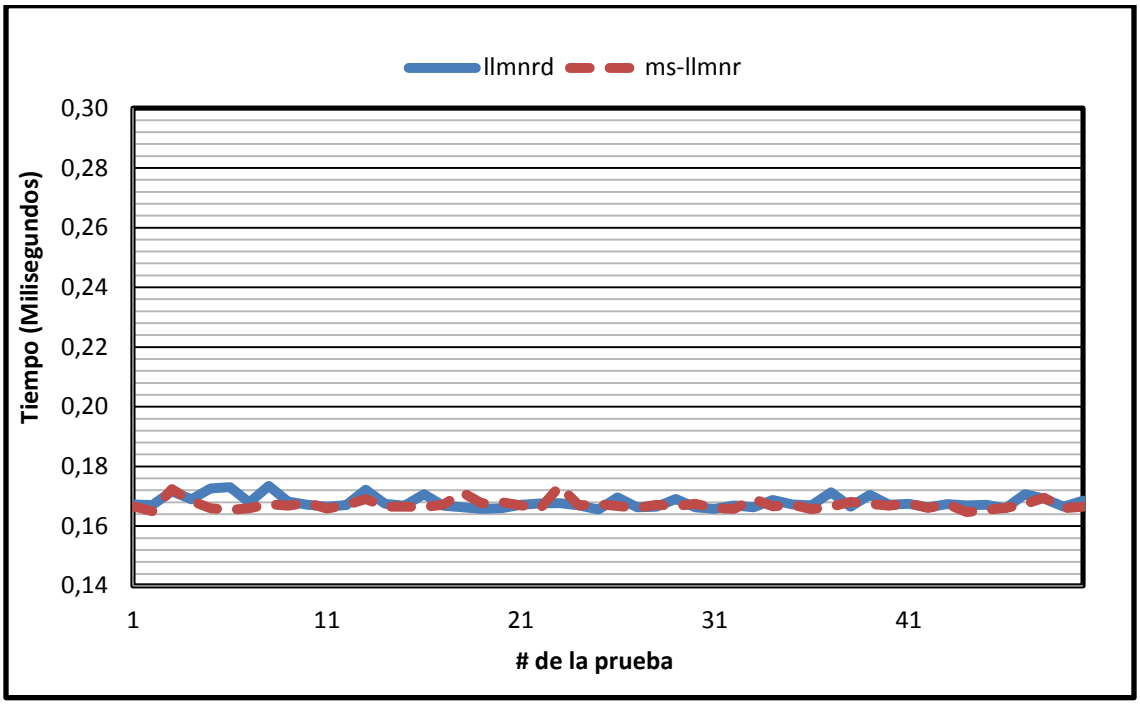


Figura 8.36: Tiempo de Respuesta Promedio (Método 1) en lacore05

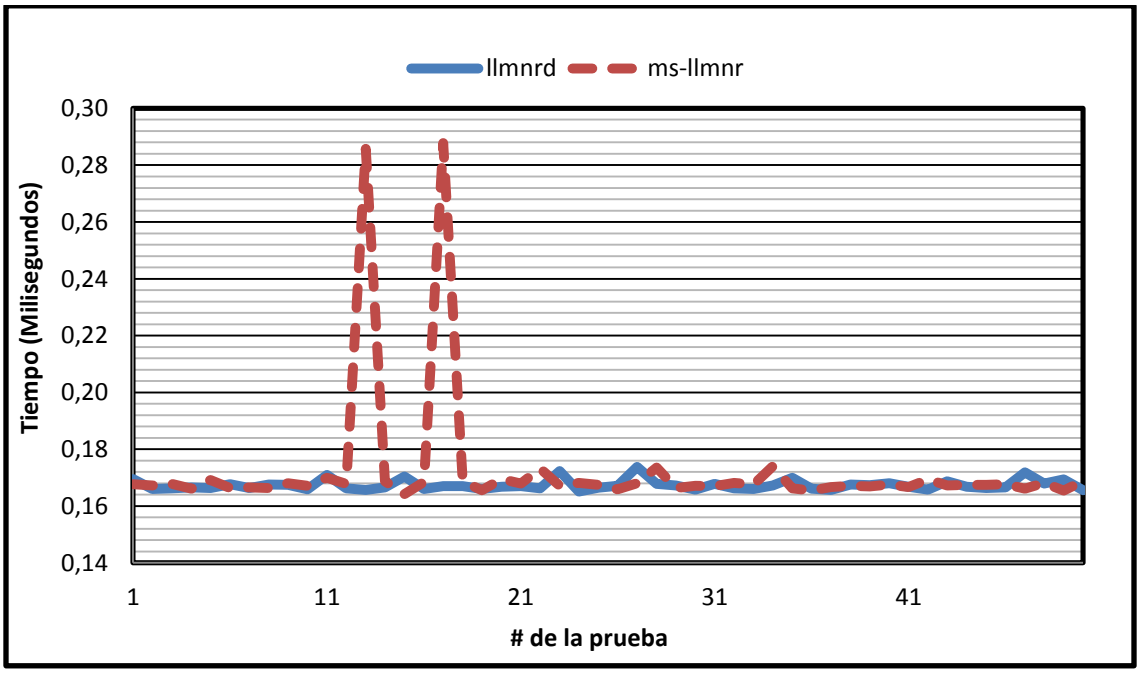


Figura 8.37: Tiempo de Respuesta Promedio (Método 2) en lacore05

Tal como puede apreciarse en las figuras citadas previamente, para el caso de *lacore04* el demonio *llmnrd* sigue teniendo un tiempo de respuesta menor, sin embargo, para el caso de *lacore05,* el desempeño de ambos demonios (*llmnrd* y *ms-llmnr*) es bastante similar.

En la Figura 8.38 y la Figura 8.39 se muestra el promedio general para el método 1 y método 2, respectivamente. El promedio general es la media aritmética del número de corridas (x=50).

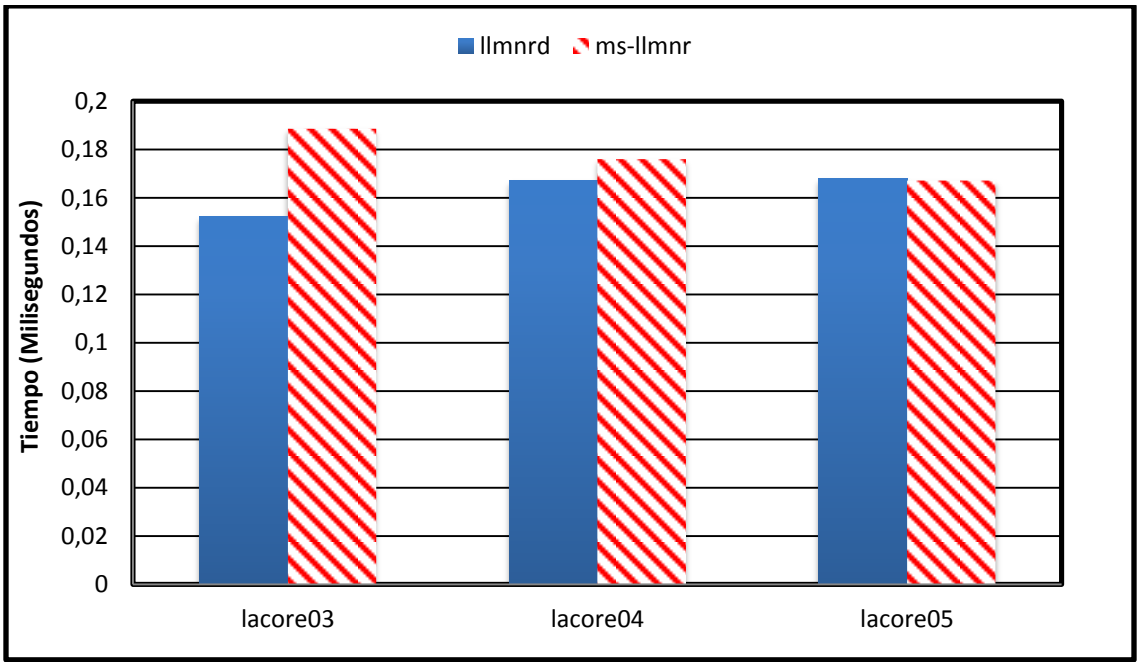


Figura 8.38: Tiempo de Respuesta Promedio General (Método 1)

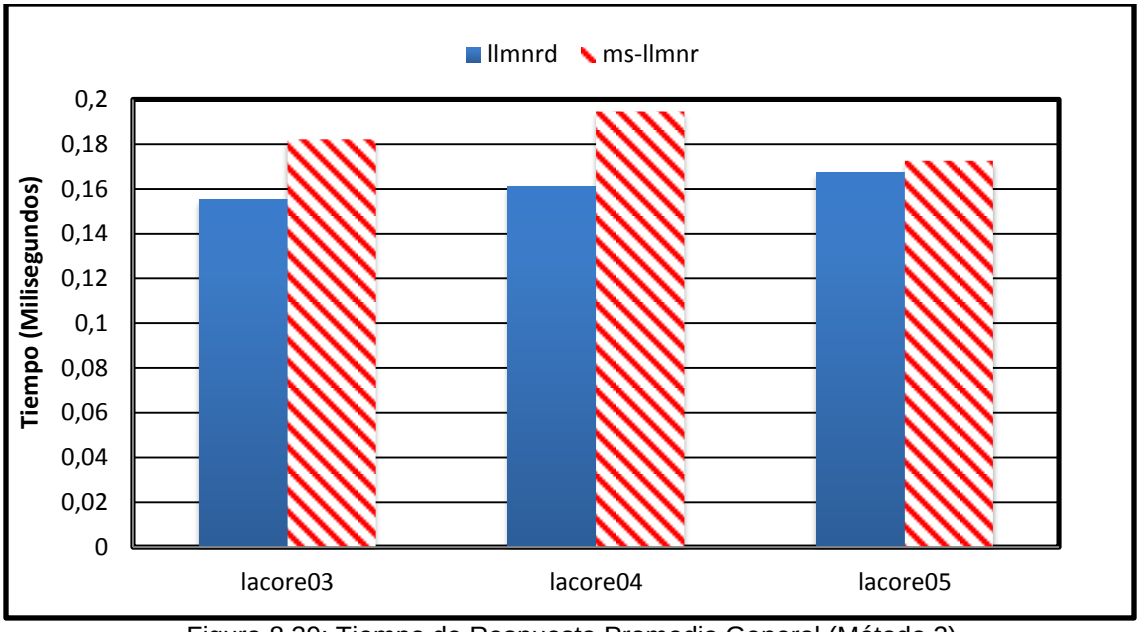


Figura 8.39: Tiempo de Respuesta Promedio General (Método 2)

Tal como puede apreciarse, y según estas mediciones, el demonio desarrollado mostró un mejor desempeño que su contraparte, el demonio *ms-llmnr*.

8.3.2 Escenario 6: Número Máximo de Peticiones Atendidas en un Segundo

Este escenario mide el número máximo de peticiones atendidas por segundo, tanto por *llmnrd* como por *ms-llmnr.* Aunque este valor puede derivarse de las pruebas realizadas en la sección previa (calculando la frecuencia), igual se obtendrá de manera empírica por 2 razones principales:

- Es más confiable y sería una ratificación de los valores obtenidos en la prueba anterior
- El código para realizar la prueba ya está implementado en la herramienta de medición, por lo que el costo de la prueba podría considerarse mínimo.

Cada petición o consulta se genera de acuerdo a las siguientes directrices:

- Se selecciona, de manera aleatoria, el valor del campo *ID* (ver Figura 4.1)
- El envío es secuencial, es decir, la petición *i* se envía enseguida despues de recibir la respuesta de la petición *i -1.*
- Una respuesta se espera hasta un tiempo máximo *LLMNR_TIMEOUT* (100 ms sugeridos en RFC 4795 [**3**]). Una vez transcurrido ese tiempo se considera que no hubo respuesta.

Para esta prueba, está disponible en la herramienta de medición el siguiente conjunto de parámetros adicionales de configuración:

- **t**: Tiempo de duración
- **x**: Número de corridas a ejecutar, es decir, cuantas veces se va a ejecutar la prueba sobre la máquina objetivo.

Comando(s) de la prueba: **mtool –c –t 1 –x 50**
Máquina *objetivo*: ***lacore03, lacore04, lacore05***

La razón de usar 3 máquinas *objetivos* para esta prueba es la misma razón presentada en la prueba anterior.

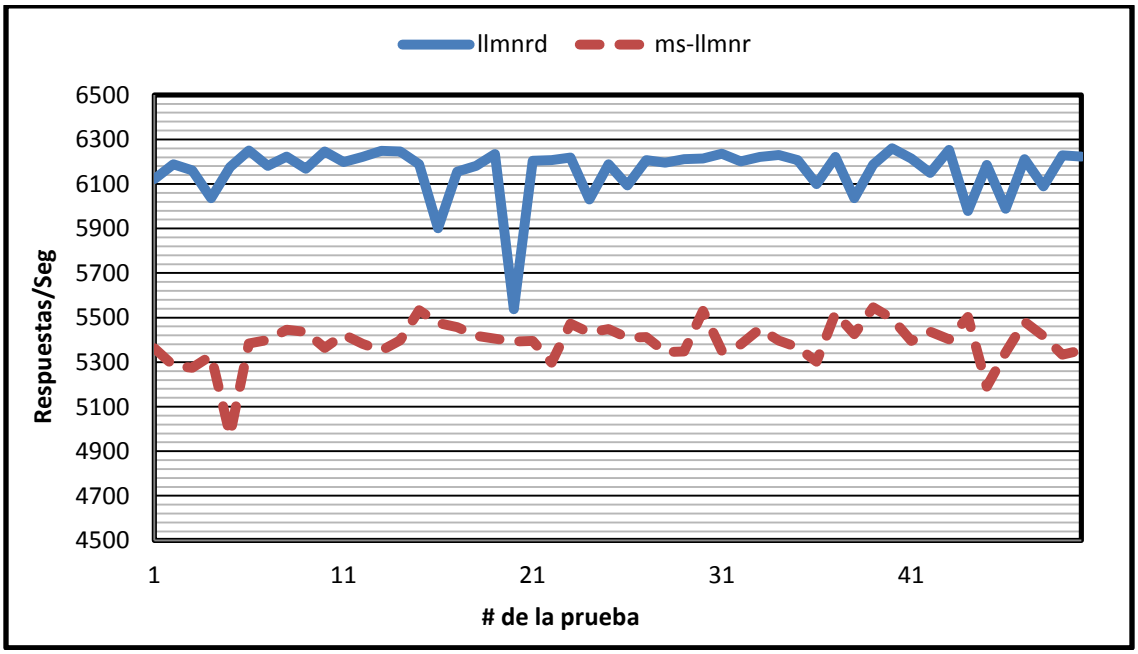


Figura 8.40: Número de Peticiones Atendidas por Segundo (lacore03)

Tal como se puede apreciar en la Figura 8.40, el demonio *llmnrd* pudo atender más peticiones por segundo que su contraparte *ms-llmnr.* Más precisamente, el demonio *llmnrd* logró contestar, en promedio general, 6161 peticiones, contra 5393 peticiones atendidas por *ms-llmnr*, lo que representa, aproximadamente, un 12% de diferencia en desempeño.

En la Figura 8.41 y la Figura 8.42 se muestra las mediciones para las máquinas *lacore04* y *lacore05*, respectivamente.

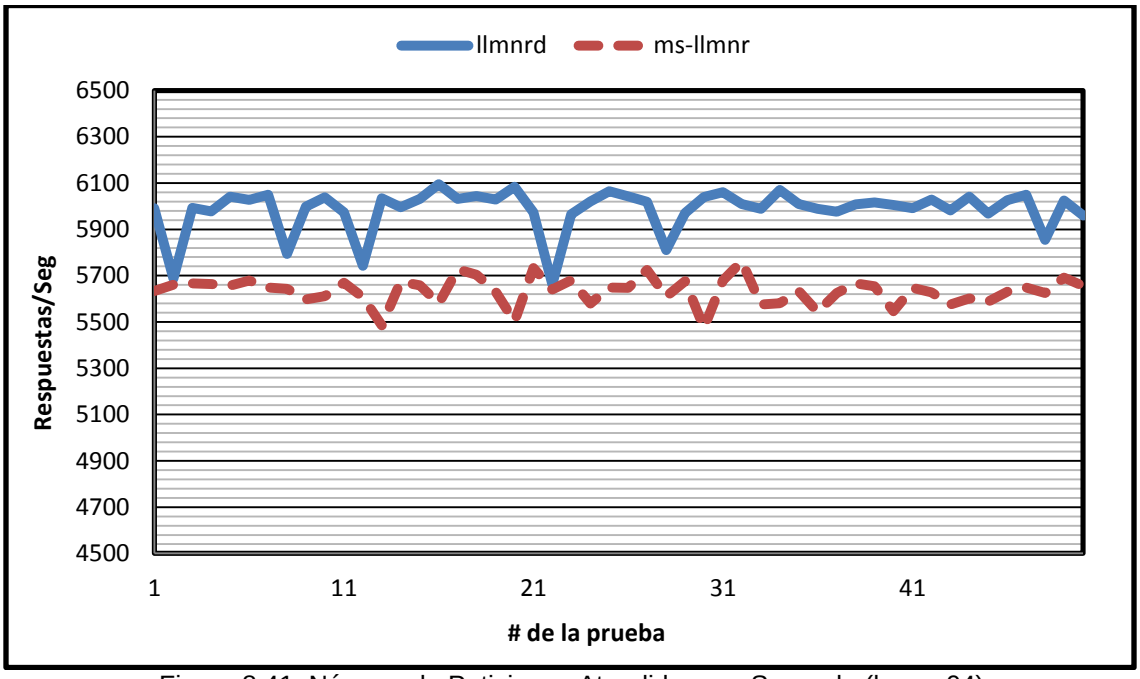


Figura 8.41: Número de Peticiones Atendidas por Segundo (lacore04)

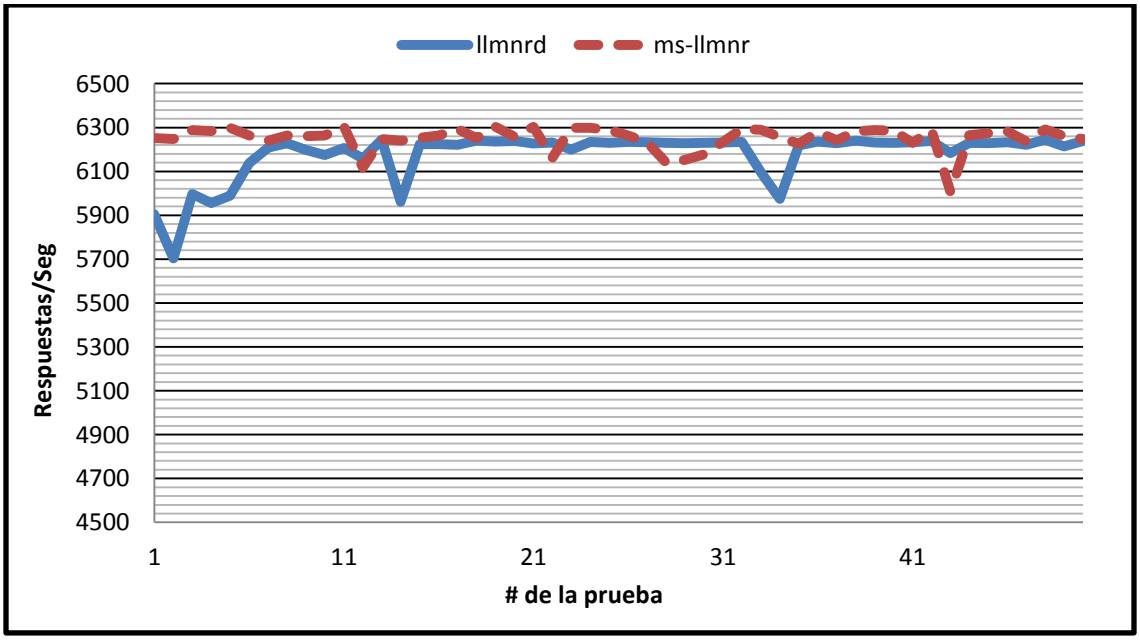


Figura 8.42: Número de Peticiones Atendidas por Segundo (lacore05)

Para el caso de *lacore04* el demonio *llmnrd* mantiene su ventaja. Para el caso de *lacore05,* el desempeño de ambos demonios es bastante más parejo, teniendo *ms-llmnr* un desempeño ligeramente superior.

En la Figura 8.43 se muestra, a manera de resumen, el promedio general para ambos demonios sobre las máquinas *lacore03, lacore04* y *lacore05.* El promedio general es la media aritmética del número de corridas (x=50).

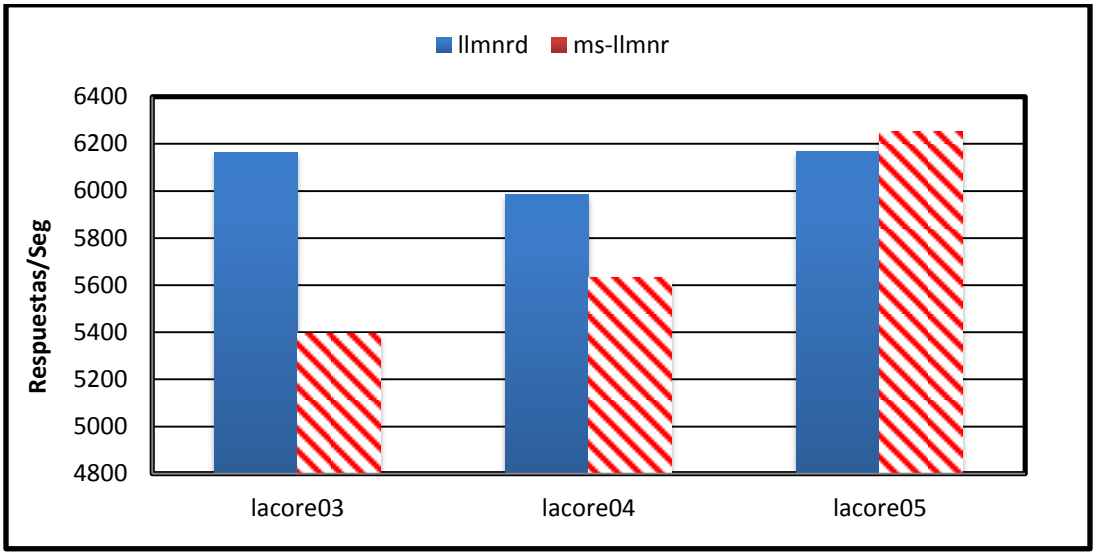


Figura 8.43: Número de Peticiones Atendidas por Segundo (Promedio General)

En la Tabla 8.1 y en la Tabla 8.2 se encuentra el resumen de los valores obtenidos en la prueba de número peticiones por segundo. También se encuentra los valores esperados derivados del cálculo de la frecuencia. En algunos casos puede apreciarse lo que podría considerarse como una diferencia pequeña (la mayor diferencia de lo obtenido versus lo esperado es de 4.8%), que se considera aceptable dado que:

- Todos los valores son promedios.
- La multitud de factores y variables que afectan un sistema computacional haciendo que su rendimiento fluctúe en el tiempo.

Máquina	Tiempo de Respuesta (Método 1)	Tiempo de Respuesta (Método 2)	Frecuencia (Método 1)	Frecuencia (Método 2)	Peticiones por Segundo Atendidas
lacore03	0.152	0.155	6578	6451	6161
lacore04	0.167	0.161	5988	6211	5985
lacore05	0.167	0.167	5988	5988	6168

Tabla 8.1: Número de Peticiones y Cálculo de la Frecuencia (llmnrd)

Máquina	Tiempo de Respuesta (Método 1)	Tiempo de Respuesta (Método 2)	Frecuencia (Método 1)	Frecuencia (Método 2)	Peticiones por Segundo Atendidas
lacore03	0.188	0.182	5319	5494	5393
lacore04	0.176	0.194	5681	5154	5633
lacore05	0.167	0.172	5988	5813	6251

Tabla 8.2: Número de Peticiones y Cálculo de la Frecuencia (ms-llmnr)

8.4 Pruebas de Estrés

En esta sección se presentan los resultados obtenidos en los diferentes escenarios de pruebas de estrés. Las pruebas ejecutadas son las siguientes:

- Prueba de Estrés 1.
- Prueba de Estrés 2.
- Tiempo de recuperación.

8.4.1 Escenario 7: Prueba de Estrés 1

Esta prueba consiste en generar una carga fuerte de trabajo con el fin de agotar, o intentar generar una denegación de servicio sobre los demonios, y al mismo tiempo medir la cantidad de fatiga presente.

Para esta prueba se introduce un nuevo rol en la herramienta de medición:

- **Atacante (attacker)**: Una máquina que inundara de peticiones LLMNR a una máquina *objetivo*. Una o varias máquinas pueden tomar este rol. Se usa el archivo de nombre *names2,* ubicado en el mismo directorio donde se encuentra el ejecutable de la herramienta de medición (*mtool*), para especificar el nombre de la(s) máquina(s) *atacante*(s).

Actuando en conjunto, una o más máquinas *atacantes,* se encargaran de inundar de peticiones a la máquina objetivo con el fin de poner estrés sobre la misma. Al mismo tiempo, o de manera simultánea, la máquina *principal*, con el fin de verificar que tanto ha sido afectada la máquina *objetivo*, se encargara de realizar una prueba de rendimiento (sobre la máquina *objetivo*, por supuesto). En este caso, la prueba de medición del rendimiento que se ejecutará es la cantidad de peticiones atendidas por segundo (especificada en la Sección 8.3.2)

Con el fin de controlar la prueba y el grado de estrés que se genera sobre un demonio, se incorporó en la herramienta de medición el siguiente conjunto de parámetros adicionales:

- **t**: Establece el tiempo (en segundos) que durará la prueba en la máquina *atacante*.
- **z**: Establece el intervalo de tiempo (en milisegundos) que la máquina *principal* deberá esperar para activar cada máquina *atacante.* La(s) máquinas(s) atacantes(s) se activarán una a una cada *z* segundos. La selección se realizará en el mismo orden especificado en el archivo mencionado previamente.
- **q**: Este parámetro establece el tiempo total (en milisegundos) durante el cual una máquina *atacante* realizará un ataque. Cabe destacar que este tiempo es común para todas las máquinas atacantes.
- **k**: Establece el intervalo de tiempo (en nanosegundos) que una máquina *atacante* debe esperar entre peticiones LLMNR. Variando este parámetro se puede controlar el grado de estrés que se genera.
- **D**: Esta opción se usa para establecer o designar una máquina como *atacante*.

Como se dijo previamente, la máquina *principal* se encarga de controlar la prueba, eso implica que también controla el comportamiento de las máquinas *atacantes* a través del intercambio de mensajes de control definidos. A continuación se listan los códigos seleccionado para identificar cada mensaje de control:

- **0x803E**: Identifica el mensaje de control de inicio de ataque.
- **0x803F**: Identifica el mensaje de control de fin de ataque.
- **0x903E**: Identifica el mensaje de control para asentir el inicio de ataque (ack).
- **0x903F**: Identifica el mensaje de control asentir el fin de ataque (ack).

La Figura 8.44, la Figura 8.45, la Figura 8.46 y la Figura 8.47 muestran el contenido completos de los distintos mensajes de control intercambiados durante esta prueba.

n bytes	2 bytes	m bytes	4 bytes
ATKERNAME	0x803E	OBJNAME	ATKTIME (-k)

Figura 8.44: Mensaje de Control (Inicio de Ataque)

2 bytes	n bytes
0x903E	ATKERNAME

Figura 8.45: Mensaje de Control (Ack Inicio de Ataque)

n bytes	2 bytes
ATKERNAME	0x803F

Figura 8.46: Mensaje de Control (Fin de Ataque)

2 bytes	n bytes
0x903F	ATKERNAME

Figura 8.47: Mensaje de Control (Ack Fin de Ataque)

Es importante mencionar que una máquina *atacante* escuchará peticiones (mensajes de control) en la dirección multicast 224.0.0.254 / FF02::1:4 y en el puerto 17283.

Comando(s) de la prueba: **./lmtool –d –t 28 –z 3000 –q 15000 –k X** (k varía entre 100000 y 250000 nanosegundos)
Máquina *objetivo*: ***lacore05***.
Máquina(s) *atacantes:* ***lacore02, lacore03, lacore04***.

El siguiente conjunto de gráficas muestra el resultado de la prueba:

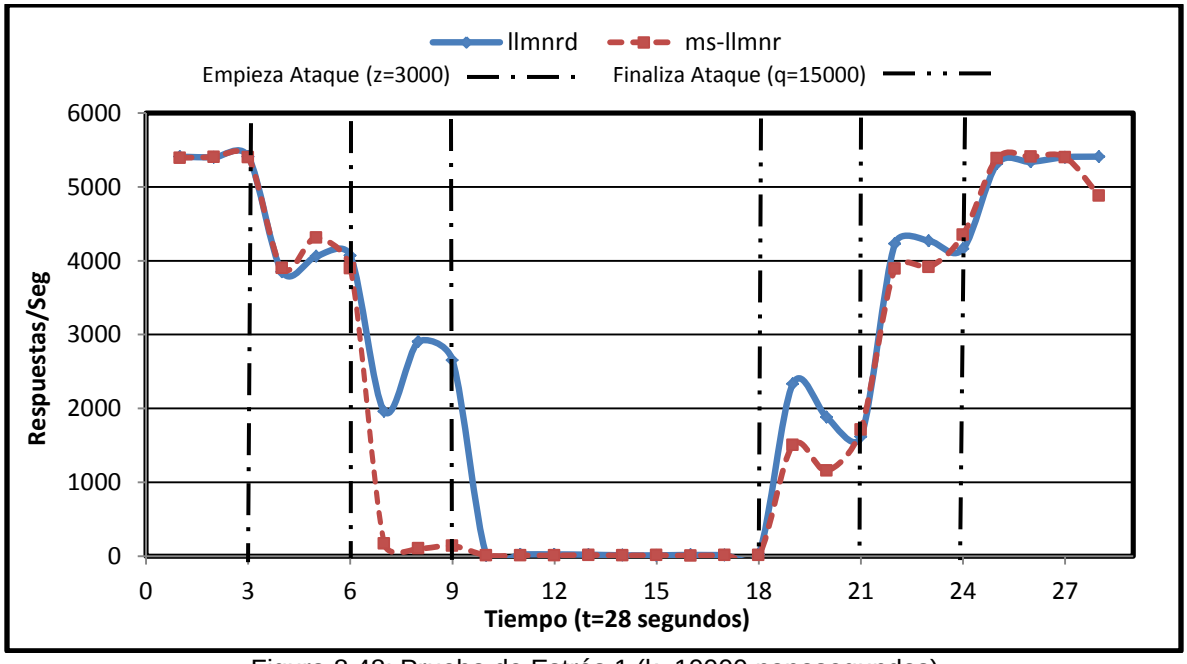


Figura 8.48: Prueba de Estrés 1 (k=10000 nanosegundos)

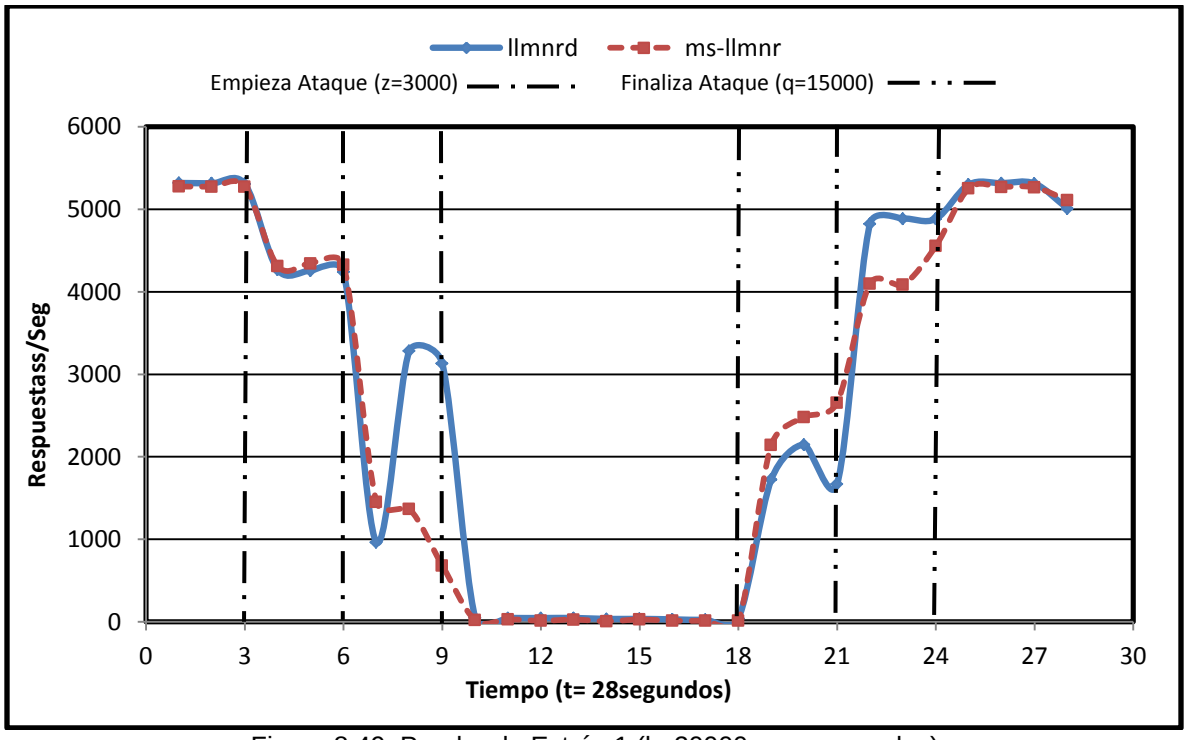


Figura 8.49: Prueba de Estrés 1 (k=20000 nanosegundos)

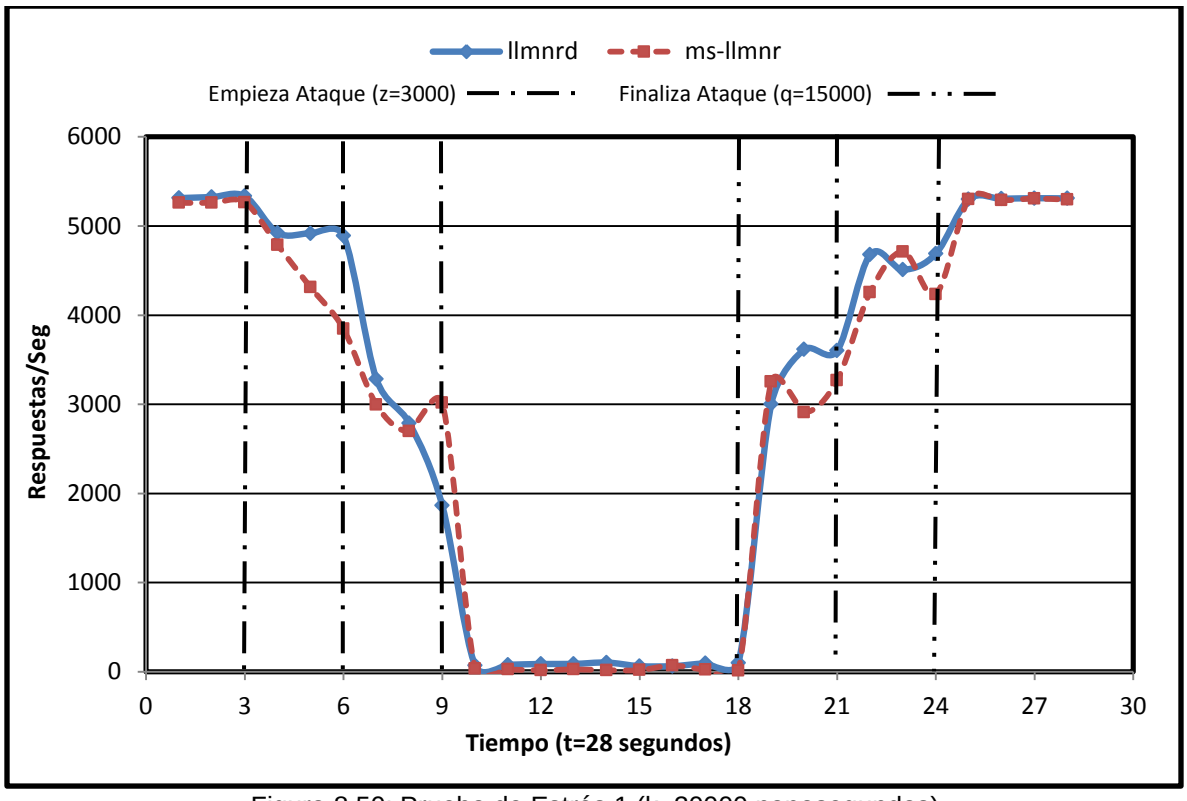


Figura 8.50: Prueba de Estrés 1 (k=30000 nanosegundos)

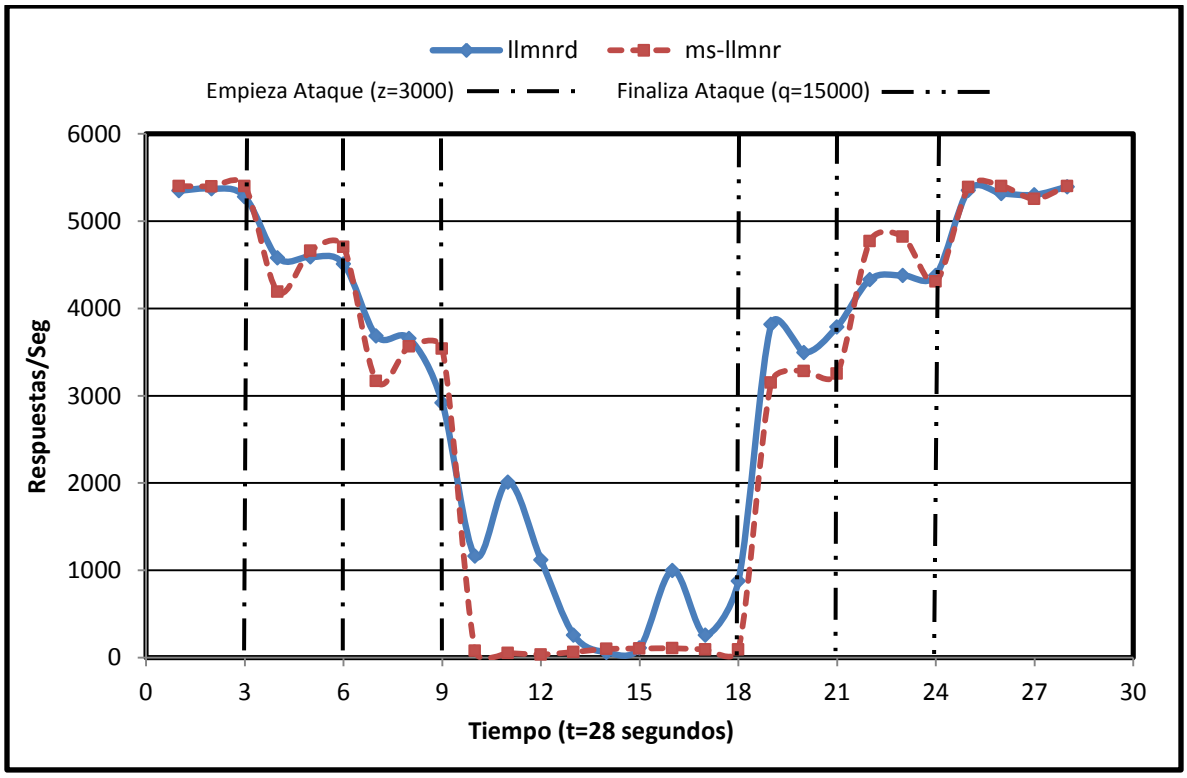


Figura 8.51: Prueba de Estrés 1 (k=40000 nanosegundos)

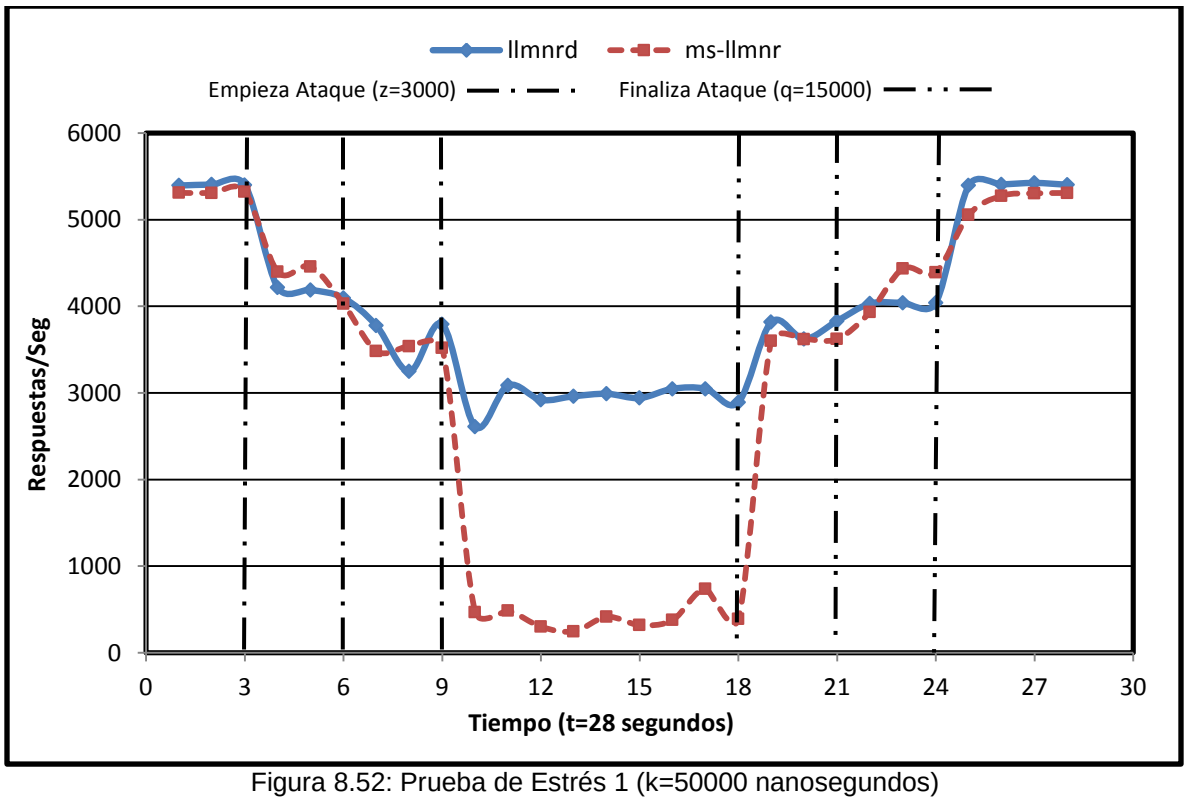


Figura 8.52: Prueba de Estrés 1 (k=50000 nanosegundos)

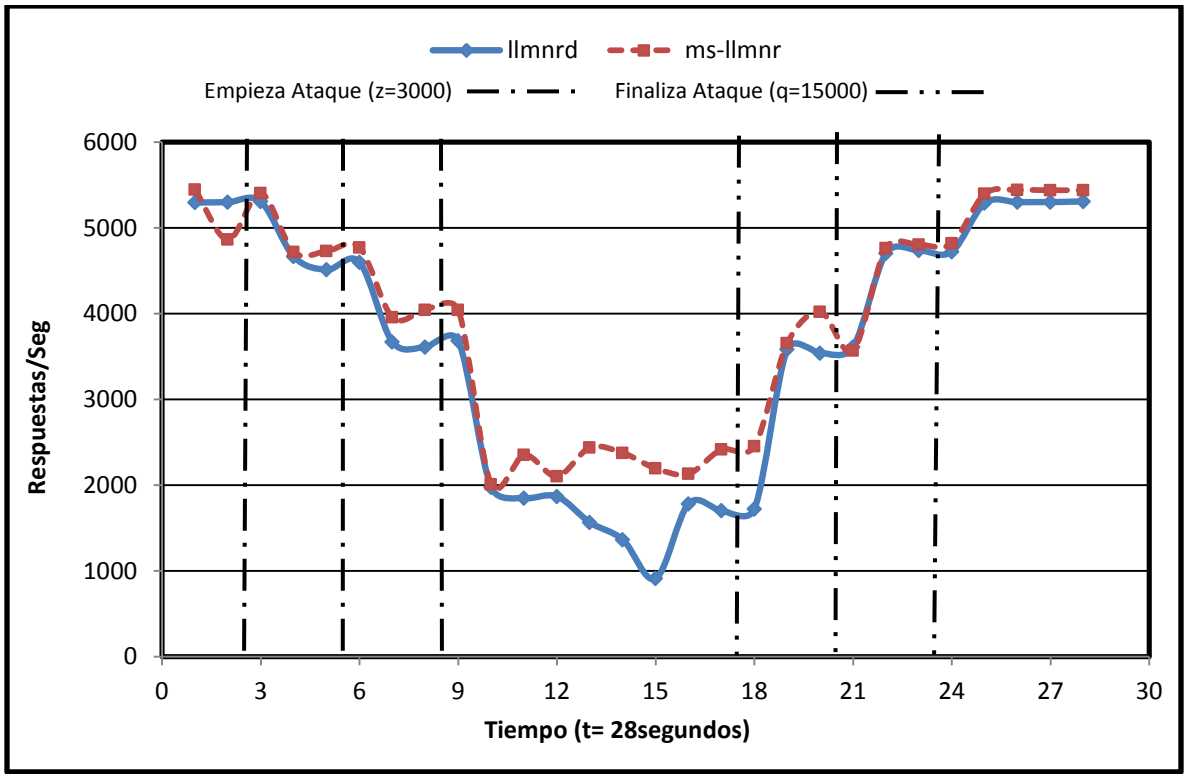


Figura 8.53: Prueba de Estrés 1 (k=75000 nanosegundos)

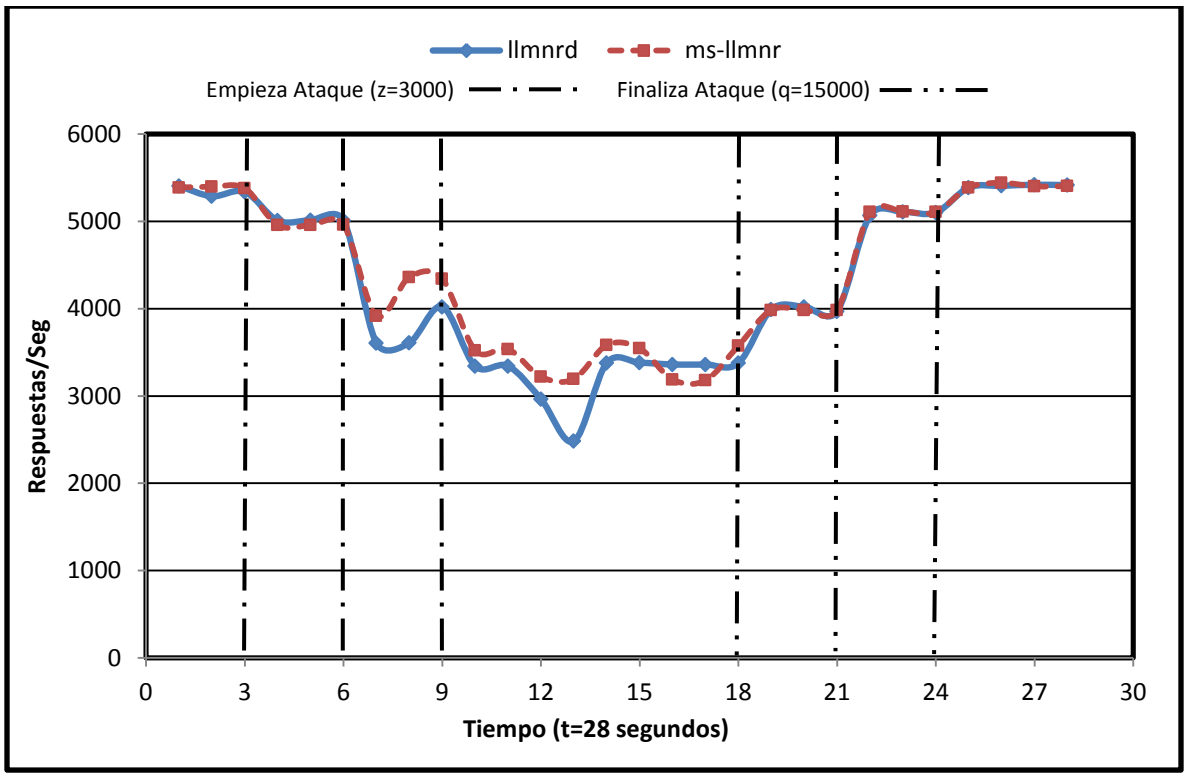


Figura 8.54: Prueba de Estrés 1 (k=100000 nanosegundos)

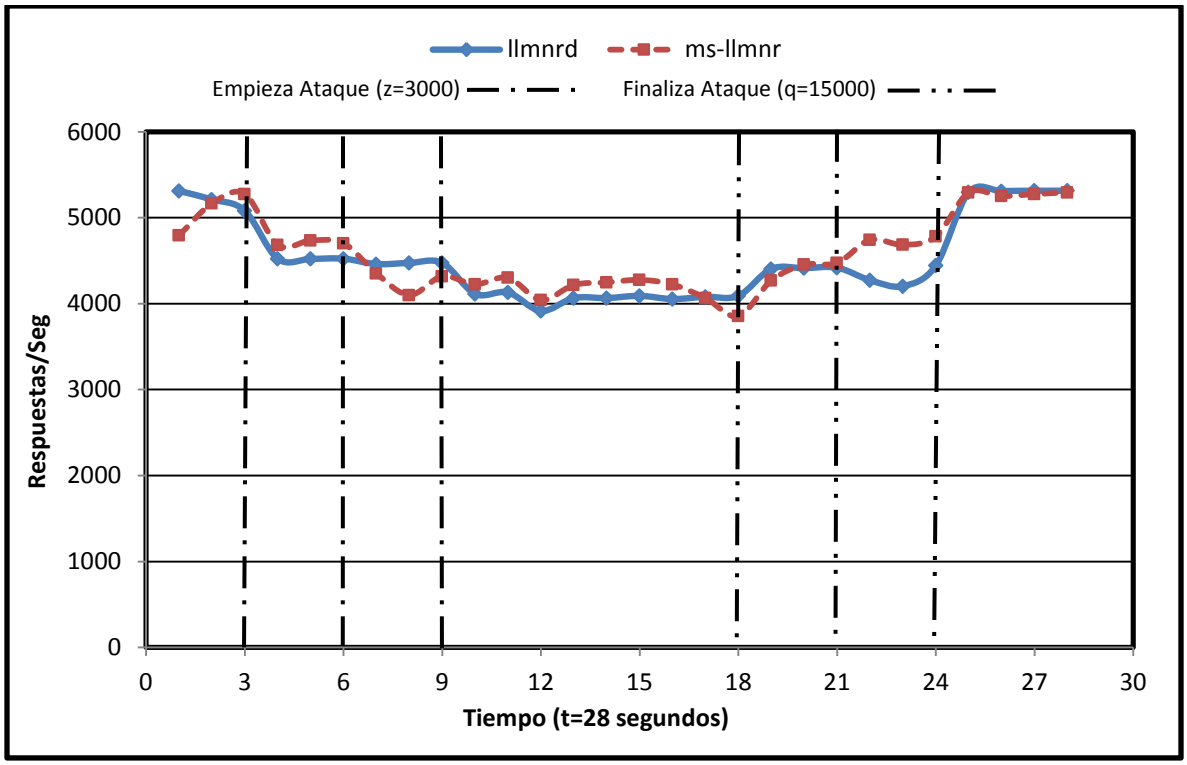


Figura 8.55: Prueba de Estrés 1 (k=150000 nanosegundos)

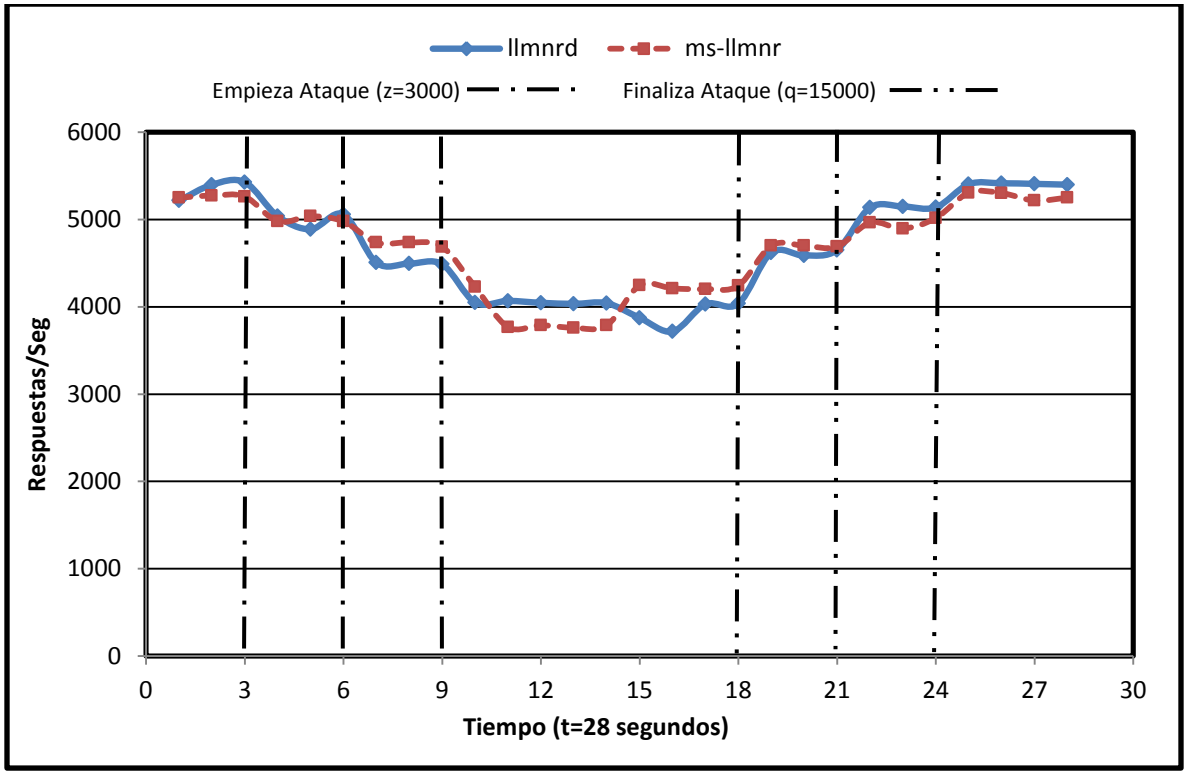


Figura 8.56: Prueba de Estrés 1 (k=200000 nanosegundos)

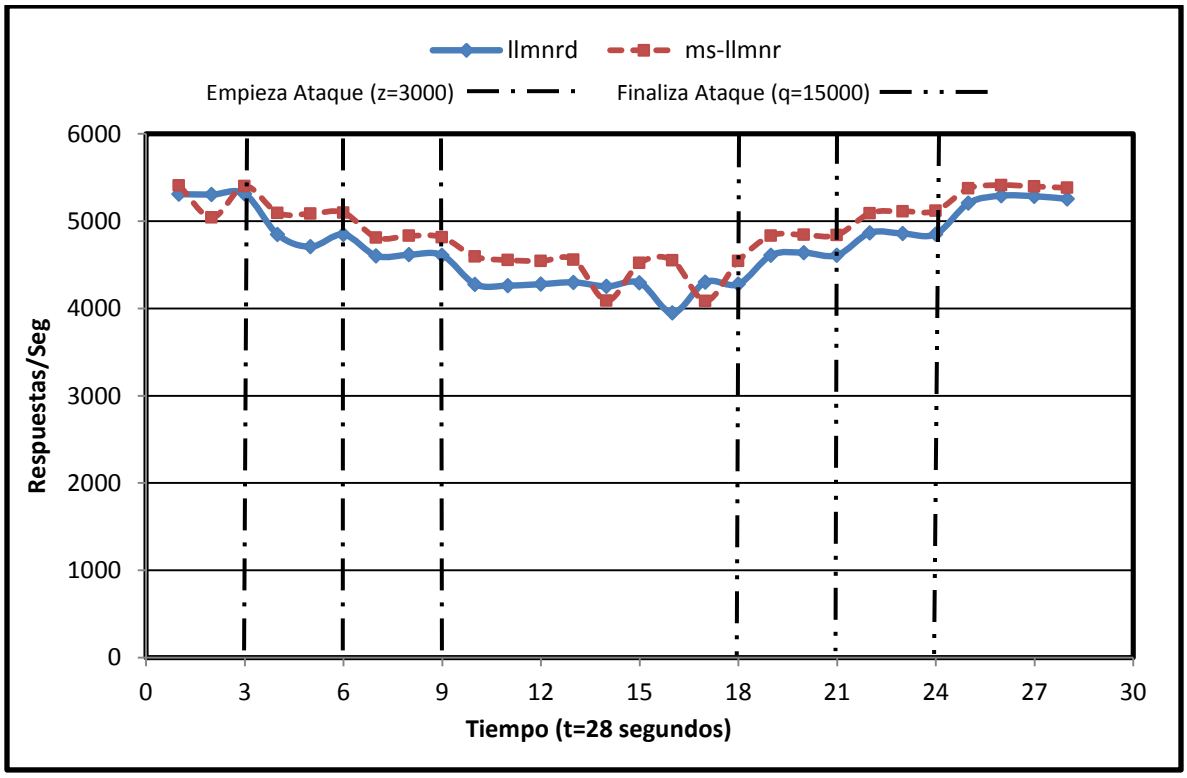


Figura 8.57: Prueba de Estrés 1 (k=250000 nanosegundos)

Como se aprecia en la Figura 8.48, Figura 8.49, Figura 8.50, Figura 8.51, Figura 8.52, Figura 8.53, Figura 8.54, Figura 8.55, Figura 8.56 y Figura 8.57, cada prueba tiene una duración total de 28 segundos, se usa un total de 3 máquinas *atacantes*, donde cada máquina *atacante* es activada cada 3 segundos (comenzando la primera a atacar en el segundo 3, la segunda en el segundo 6 y la tercera en el segundo 9), y el tiempo de ataque en cada máquina es igual a 15 segundos (finalizando el primer ataque a los 18 segundos, el segundo a los 21 segundos y el tercero finaliza en el segundo 24). El intervalo de variación entre peticiones de la máquina *atacante* comienza en 10000 y va aumentando con una razón aritmética de 10000, 25000 y 50000 nanosegundos. En cuanto al comportamiento en estas pruebas, puede decirse que ambos demonios tuvieron un rendimiento relativamente similar, de hecho no hay mucho que pueda decirse que no sea de apreciación trivial en las gráficas previamente citadas.

8.4.2 Escenario 8: Prueba de Estrés 2

Esta prueba es bastante similar a la *prueba de estrés 1* (Sección 8.4.1). La diferencia radica en pausar la tarea de las máquinas *atacantes* (es decir pausar el ataque) durante un instante de tiempo determinado. A continuación se mencionan un conjunto de características relativas a esta prueba:

- La pausa y la correspondiente reactivación de las máquinas *atacantes* es controlada por la máquina *principal*
- La pausa del ataque comienza solo después que todas y cada una de las máquinas *atacantes* hayan sido activadas.
- La pausa es genérica, es decir, una vez dictaminada una pausa esta aplica para todas las máquinas *atacantes*.
- El tiempo de pausa puede tomar un valor entre 0 y 1000ms ($0 < pause < 1000$).

Para esta prueba está disponible en la herramienta de medición el mismo conjunto de parámetros adicionales de configuración disponibles para la prueba de estrés 1 (Sección 8.4.1). Adicionalmente esta disponible el siguiente conjunto de parámetros de configuración:

- **n**: Esta opción se usa para establecer el tiempo parcial (en milisegundos) que durará un ataque antes de una pausa. Más específicamente, el tiempo de pausa sería el complemento: 1000 – n (como se dijo anteriormente el tiempo de pausa toma un valor entre 0 y 1000 milisegundos

Para esta prueba se agregan 2 mensajes de control extras al intercambio. A continuación se listan los códigos asignados a estos mensajes de control:

- **0x804A**: Identifica el mensaje de control de pausa de ataque.
- **0x804B**: Identifica el mensaje de control de reanudar ataque.

La Figura 8.58 y la Figura 8.59 muestran el contenido de los 2 mensajes de control nuevos para esta prueba.

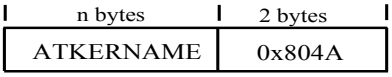


Figura 8.58: Mensaje de Control (Pausa Ataque)

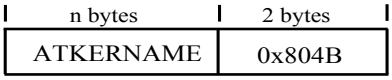


Figura 8.59: Mensaje de Control (Reanudar Ataque)

En la Figura 8.60 y a manera de ejemplo ilustrativo se muestra la distribución de los tiempos que maneja la máquina *principal* para una *prueba* de estrés *2* (hipotética), con 3 máquinas *atacantes*, t=9segs, q=5000ms, z=1000ms y n=400ms.

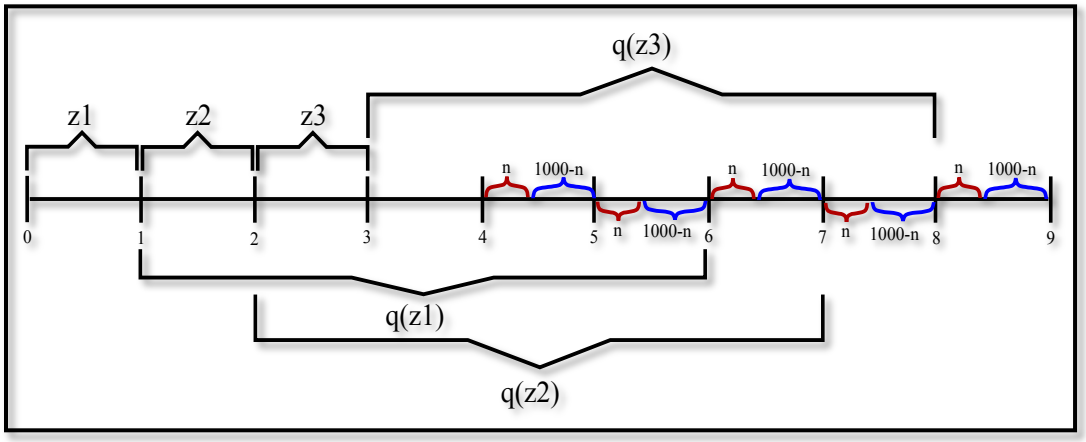


Figura 8.60: Distribución de los Tiempos Manejados en Máquina Principal

Este ejemplo es pertinente puesto que ayuda a aclarar dudas que puedan generarse respecto al significado de los distintos parámetros asociados al tiempo, manejados por una máquina *principal* en esta prueba (también aplica para la prueba de estrés 1). A continuación se describe los eventos ocurridos en el tiempo, que se derivan de la prueba hipotética, previamente mencionada:

- Al segundo 1, 2 y 3 se envía un mensaje de control (inicio de ataque) dirigido, respectivamente, a la máquina *atacante* 1, 2 y 3.

- Al segundo 4+n, 5+n,…, 8+n se envía un mensaje de control (pausar ataque) dirigido a las máquinas *atacante* 1, 2 y 3. Es preciso aclarar que por cada intervalo de tiempo (mencionado anteriormente) se envía un único mensaje de control (en lugar de 3) con ATKERNAME="all".
- Al segundo 5 (4+n+(1–n)=4+1=5), 6, 7 y 8 se envía un mensaje de control (reanudar ataque) dirigido a las máquinas *atacante* 1, 2 y 3 (se envía un único mensaje por cada intervalo de tiempo con ATKERNAME="all").
- Al segundo 6, 7 y 8 se envía un mensaje de control (fin de ataque) dirigido, respectivamente, a las máquinas *atacante* 1, 2 y 3.
- Al segundo 9 finaliza la prueba.

De lo mencionado anteriormente se puede deducir lo siguiente:

- El mensaje de control (reiniciar ataque) enviado en el segundo 8 es un mensaje que no será tomado en cuenta por las máquinas *atacantes* (puesto que ya todas han concluido).
- Análogamente, lo mismo ocurre para el mensaje de control (pausar ataque) enviado en el segundo 8+n.
- La *prueba de estrés 1* es el equivalente a la *prueba de estrés 2* con n=∞

Comando(s) de la prueba: **./mtool –e –t 28 –z 3000 –q 15000 –k 0 –n X** (n varía entre 900 y 300 milisegundos)
Máquina *objetivo*: ***lacore05***
Máquinas *atacantes*: ***lacore02, lacore03, lacore04***.

El siguiente conjunto de gráficas muestra el resultado de la prueba:

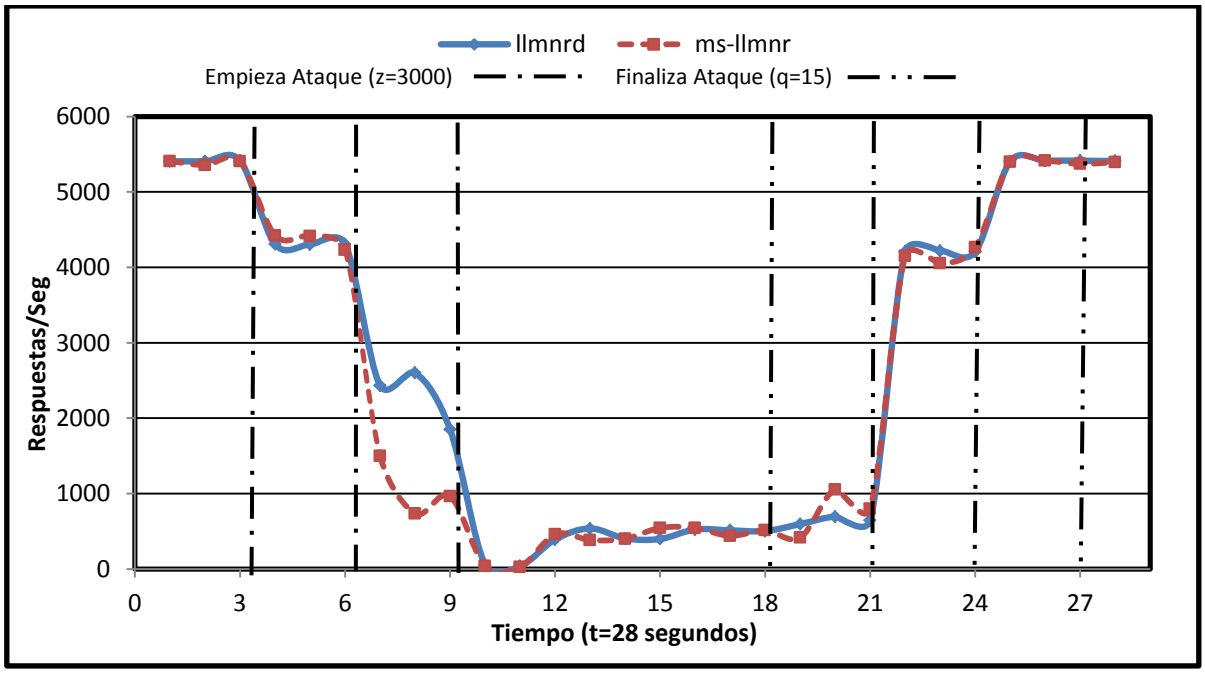


Figura 8.61: Prueba de Estrés 2 (n=900ms)

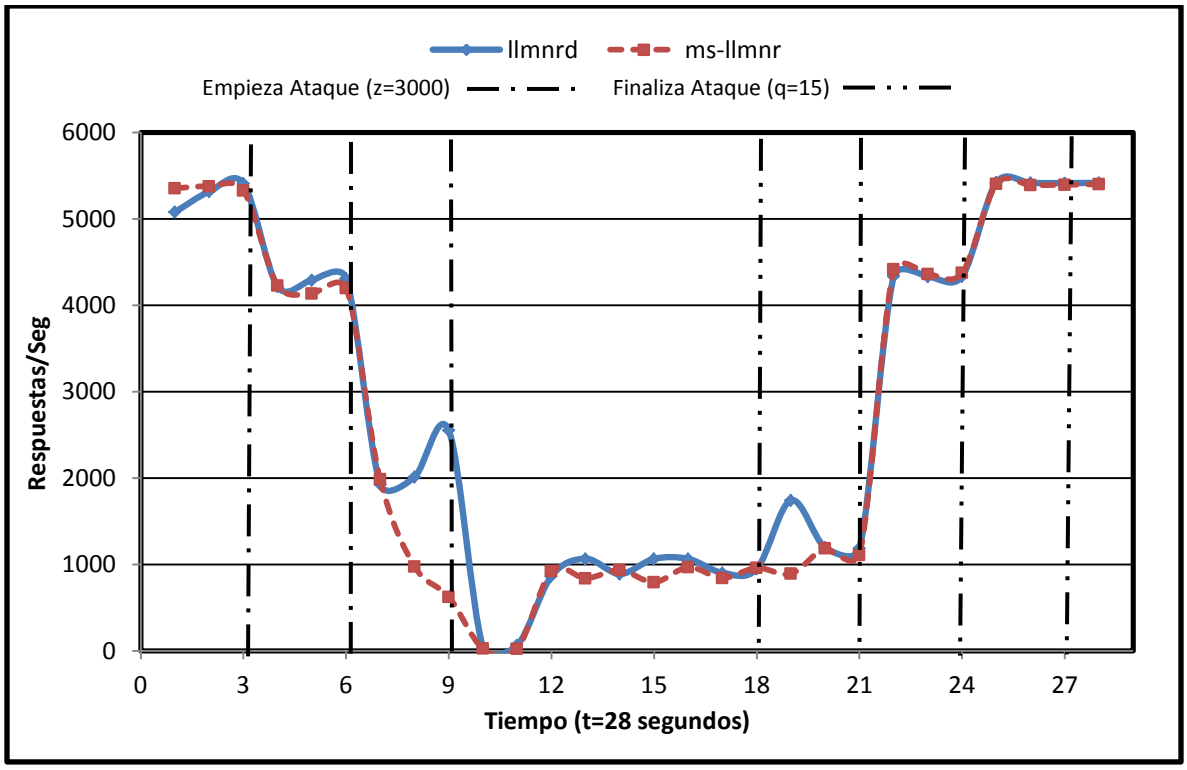


Figura 8.62: Prueba de Estrés 2 (n=800 ms)

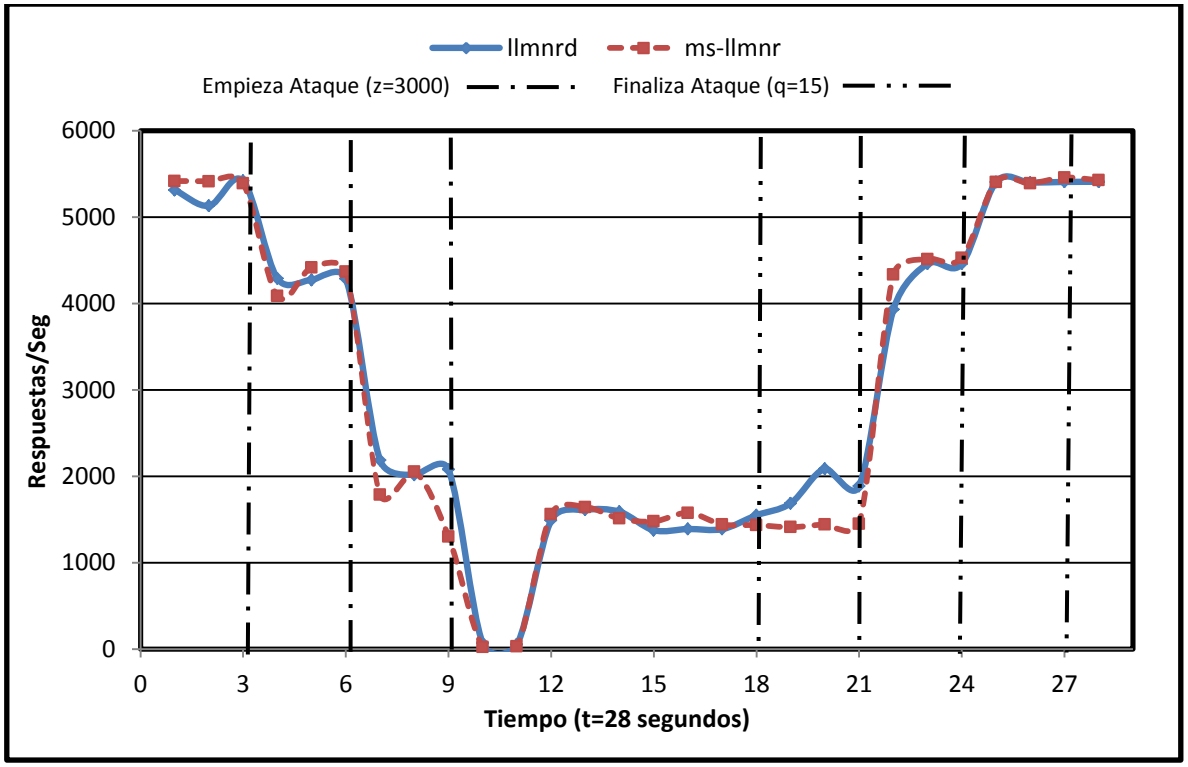


Figura 8.63: Prueba de Estrés 2 (n=700ms)

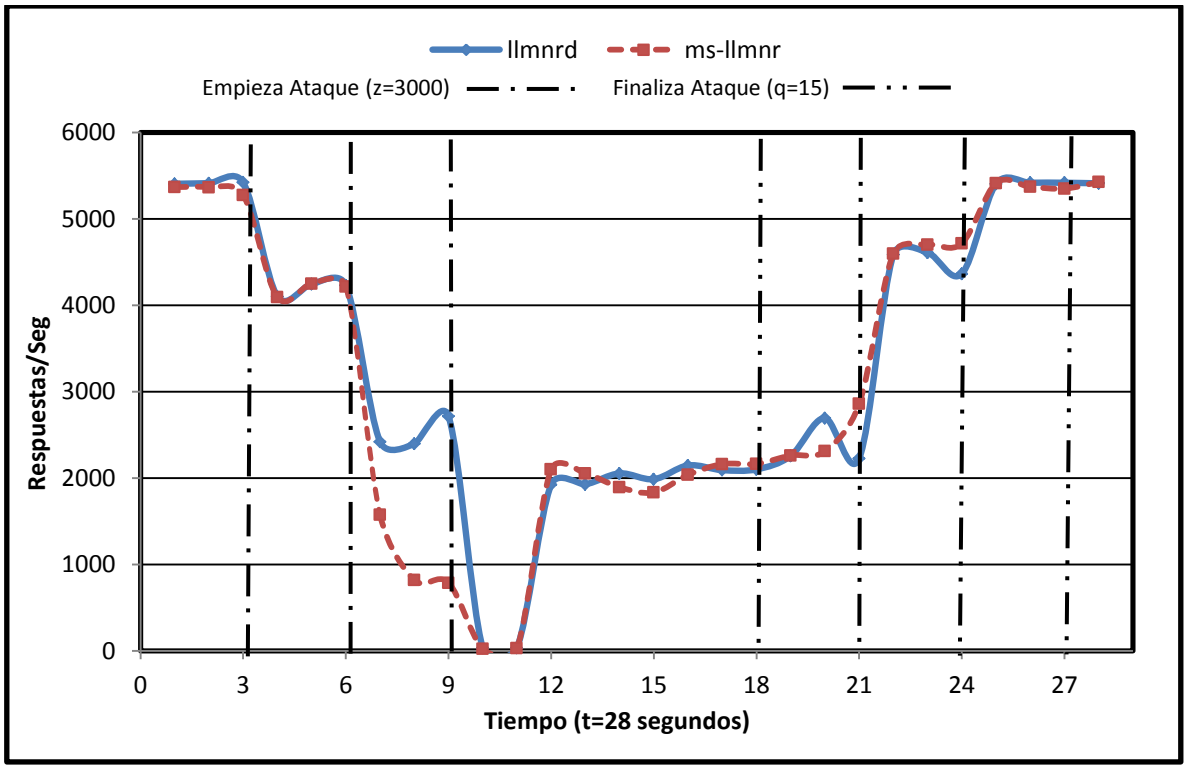


Figura 8.64: Prueba de Estrés 2 (n=600ms)

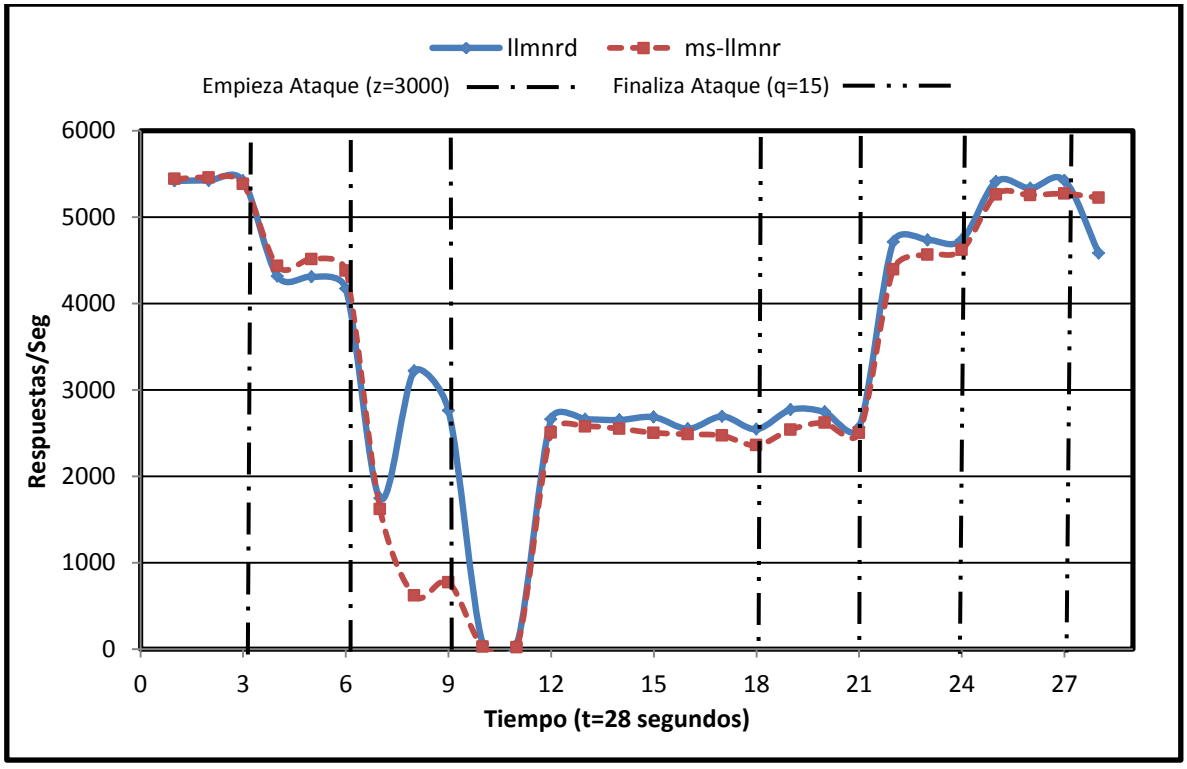


Figura 8.65: Prueba de Estrés 2 (n=500ms)

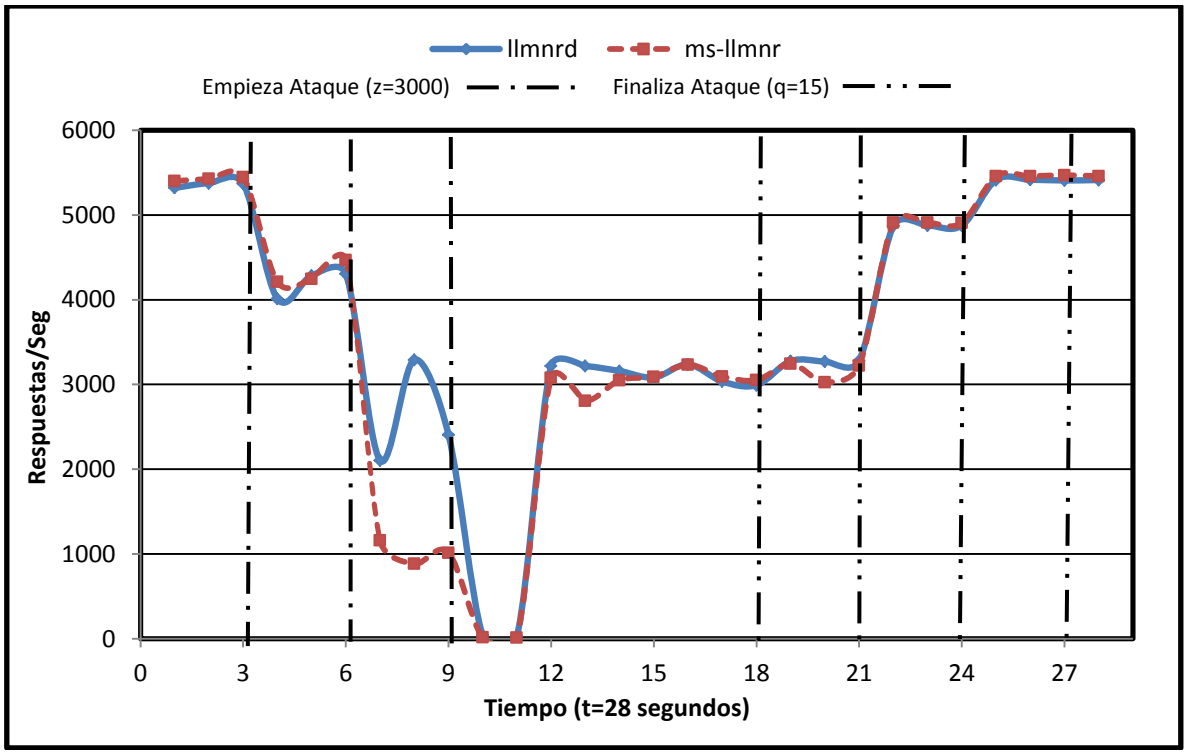


Figura 8.66: Prueba de Estrés 2 (n=400ms)

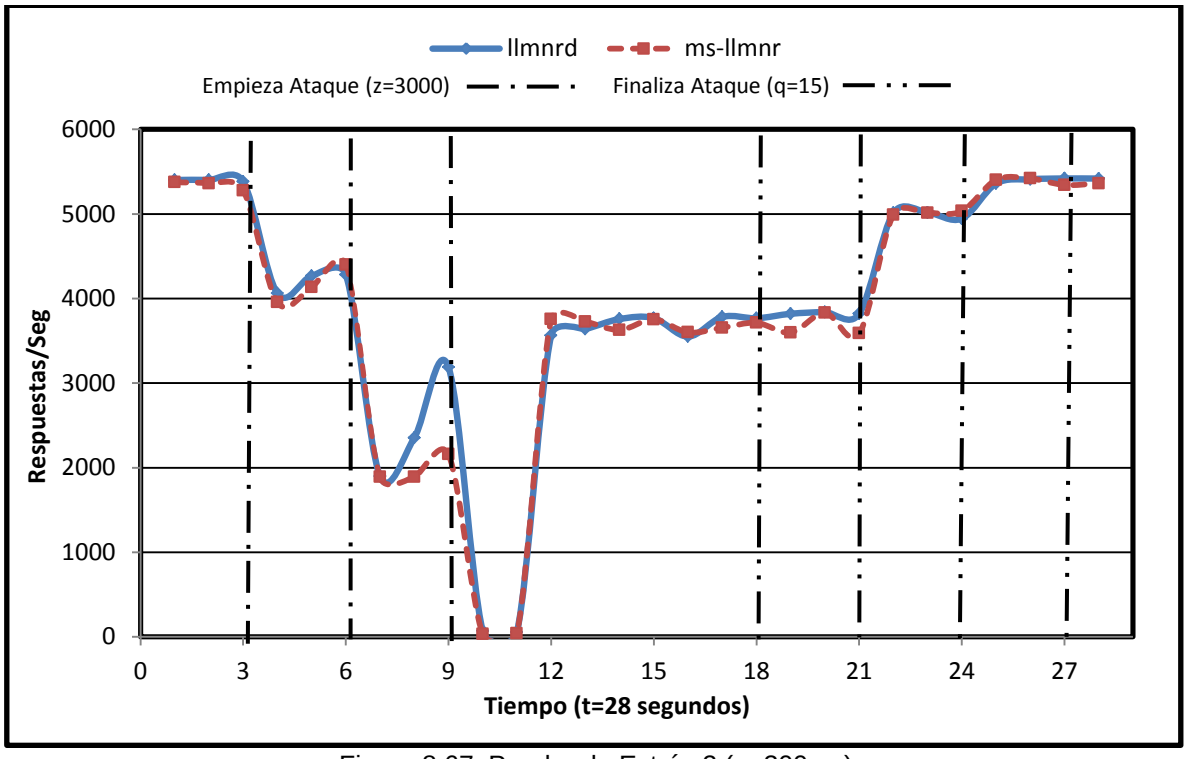


Figura 8.67: Prueba de Estrés 2 (n=300ms)

Como se puede apreciar en la Figura 8.61, la Figura 8.62, la Figura 8.63, la Figura 8.64, la Figura 8.65, la Figura 8.66 y la Figura 8.67 (y al igual que en la *prueba* de estrés 1), cada prueba tiene una duración total de 28 segundos, se usa 3 máquinas *atacantes*, (cada

máquina atacante es activada cada 3 segundos y el tiempo de ataque en cada máquina es igual a 15 segundos (finalizando el primer ataque a los 18 segundos, el segundo a los 21 segundos y el tercero finaliza en el segundo 24). El intervalo de variación de tiempo parcial de ataque comienza en 900ms y va disminuyendo con una razón aritmética de 100ms hasta alcanzar los 300ms. En las figuras previamente citadas, puede apreciarse el rendimiento similar que presentan ambos demonios. Si observamos la gráfica de la Figura 8.61, se puede apreciar una recuperación con respecto a la gráfica de la Figura 8.48 (recuérdese que la diferencia entre la prueba de estrés 1 y la *prueba de* estrés 2 es que en la primera el parámetro *n* no varía). Se infiere entonces que una pausa de 100ms es suficiente para que ambos demonios comiencen a recuperar su ritmo de respuesta.

8.4.3 Escenario 9: Tiempo de Recuperación

Esta prueba consiste en medir, una vez finalizado un ataque, de manera relativamente precisa, el tiempo de recuperación. Este tiempo de recuperación se define como el tiempo transcurrido en el que un demonio puede comenzar a responder peticiones con la misma frecuencia con que respondía previo al ataque. Para realizar esta prueba se tomará en cuenta los siguientes pasos:

- Generar la máxima carga posible de trabajo con el fin de agotar totalmente la máquina *objetivo* (el demonio que está siendo probado).
- Una vez transcurrido un tiempo determinado (bajo máximo estrés) detener en seco todas las máquinas *atacantes* que se encuentren activas al momento. El tiempo transcurrido para detener el ataque debe ser negociado previamente (entre la máquina *principal* y las *atacantes*). No se puede usar mensajes de control (la red estará inundada) para coordinar la detención súbita.
- Una vez detenido el ataque comenzar a generar, desde la máquina *principal*, peticiones en espera de una respuesta y verificar el tiempo transcurrido. El tiempo entre peticiones debe ser bajo (podría haber descarte de paquetes por la saturación de la red y/o máquina *objetivo*). Para este caso se escogió 10ms.

Para esta prueba está disponible en la herramienta de medición el siguiente conjunto de parámetros adicionales de configuración:

- **t**: Establece el tiempo de coordinación entre las máquinas atacantes y la máquina principal. Para una máquina *atacante* este tiempo es la duración del ataque. Para la máquina *principal* este es el tiempo que deberá esperar para comenzar a generar peticiones.
- **x**: Número de corridas a ejecutar.

En la Figura 8.68 se muestra el mensaje de control utilizado para esta prueba.

n bytes	2 bytes	m bytes	4 bytes
ATKERNAME	0x503E	OBJNAME	ATKTIME (-t)

Figura 8.68: Mensaje de Control (Inicio de Ataque 2)

Es importante comentar los siguientes aspectos a tener en cuenta para esta prueba:

- Se considerará como tiempo de recuperación el tiempo de llegada de la primera respuesta LLMNR válida. Esto implica que todas las peticiones generadas por la máquina *principal* deben ser almacenadas puesto que no se sabe cuáles peticiones se perderán o en qué orden serán respondidas.
- Se medirá el tiempo de llegada de una segunda y tercera respuesta (ambas válidas), con el fin de verificar la “normalización” del servicio.

- Antes de transmitir el mensaje de control (*inicio de ataque 2*) se calculará el RTT (round trip time) de la red (para hacer más preciso el tiempo de coordinación.
- Se enviará un único mensaje de control (*inicio de ataque 2*) con ATKERNAME= "all" (no se usará ack para este mensaje de control).

Comando(s) de la prueba: **./mtool –f –t 15 –x 10**.
Máquina *objetivo*: ***lacore05***.
Máquinas *atacantes*: ***lacore01, lacore02, lacore03, lacore04, lacore07***.

En la Figura 8.69 puede apreciarse los resultados de la misma.

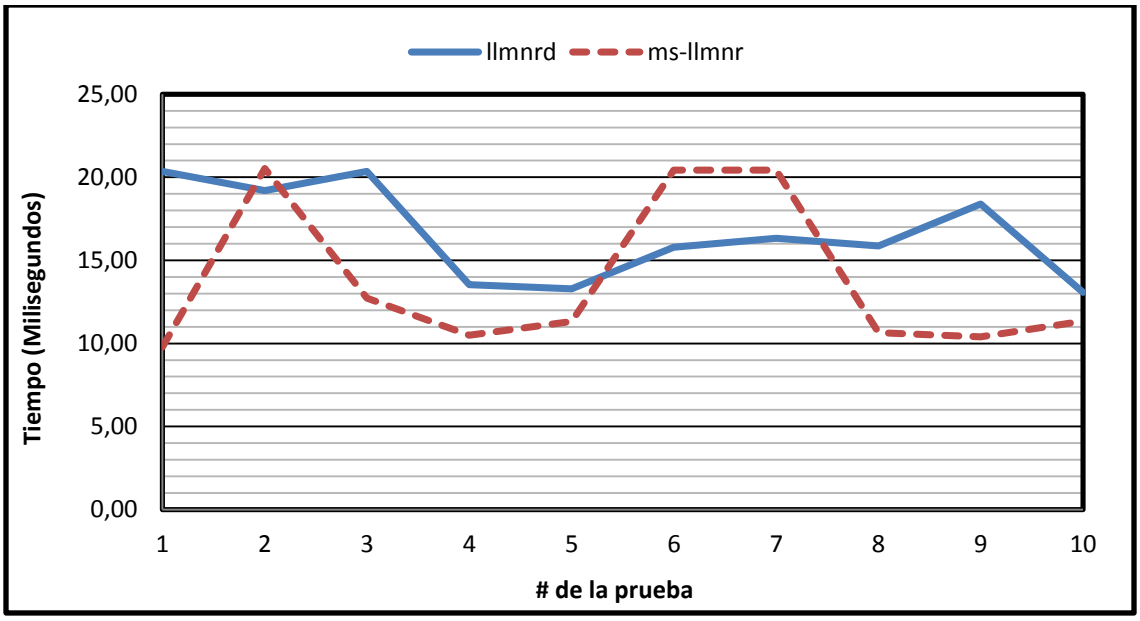


Figura 8.69: Tiempo de Recuperación

En la Tabla 8.3 puede observarse el tiempo de recuperación promedio.

Demonio	Tiempo de Recuperación (promedio)
llmnrd	16.617ms
ms-llmnr	13.813ms

Tabla 8.3: Tiempo de Recuperación (promedio)

8.5 Especificaciones Técnicas

El demonio *llmnrd* fue desarrollado para distribuciones GNU/Linux (x86 y x86_64). Su instalación se lleva a cabo a través del comando *make install,* tanto en el directorio raíz del Sender como en el directorio raíz del Responder (ambos están en directorios separados). Es necesaria la herramienta *make* y el compilador *gcc* para la instalación. En caso que *gcc* no esté presente, puede usarse cualquier compilador que soporte las banderas "MD" y "MP" (provistas por *gcc*). En ese caso habría que actualizar la variable *CC* del archivo *Makefile* (tanto del Sender como del Responder).

9. **Conclusiones y Trabajos Futuros**

LLMNR nace con el fin de cubrir las necesidades para resolver nombres de equipo dentro de una red local (LAN) de manera eficiente y automática. Si bien ya existe un protocolo robusto y bien establecido para dicha tarea, como lo es DNS, este es todo menos automático, puesto que requiere conocimientos técnicos mínimos para ponerlo en funcionamiento, cosa que para muchos usuarios podría ser una tarea engorrosa. Más aun, con el eventual dominio del protocolo IPv6 sobre las redes informáticas, y sus sobresalientes características de autoconfiguración, los protocolos de tipo "zero configuration" apuntan a ser la regla y no la excepción. Si bien, ya existe una alternativa similar al protocolo LLMNR, denominada multicast DNS (mDNS), esta opción no se encuentra soportada por la familia de sistemas operativos Microsoft Windows, que domina el mercado de computadores personales por una amplia mayoría. Si bien es perfectamente posible lograr resolver nombres de equipo de manera local, sin la necesidad de implementar un servidor DNS, entre Microsoft Windows y GNU/Linux, a través del protocolo NetBIOS, esta no es una verdadera solución, puesto que NetBIOS se encuentra oficialmente en fase de caducidad, ya que Microsoft, quien es dueña del protocolo, lo ha colocado en estado "deprecated" (uso no recomendado).

Pero yendo más allá de la necesidad de tener interoperabilidad entre estos 2 sistemas operativos, vale decir que GNU/Linux es conocido por ser un sistema operativo muy robusto, flexible y diverso (esto se debe a que su grupo de desarrolladores y colaboradores es muy grande), cuya filosofía apunta a hacer las cosas de manera correcta, eficiente y apegado a los estándares, por lo que implementar el protocolo LLMNR parece ser más una "obligación moral" que una necesidad de compatibilidad con Microsoft Windows.

En este trabajo especial de grado, se diseñó e implementó el protocolo LLMNR, para funcionar bajo el sistema operativo GNU/Linux, con soporte nativo para IPv4 e IPv6, incorporando todas las características obligatorias, y una gran mayoría de características opcionales, definidas por el estándar RFC 4795 [**3**]. También se consideró características de seguridad a la hora de implementar los distintos algoritmos que forman parte del código fuente de la implementación, que si bien no son características mencionadas en el estándar, son cuestiones inherentes al hecho de programar, como por ejemplo el correcto uso de la memoria, la correcta validación de parámetros para evitar accesos a memoria no contemplados, el manejo de data externa en memoria ubicada en el *heap, etc.* Si bien se ha mencionado en varias oportunidades que LLMNR es un protocolo que no requiere ningún tipo de configuración o mediación por parte de un usuario o administrador para funcionar, esto no implica bajo ninguna circunstancia que el tema de configuración de parámetros sea un tema de segundo grado, por lo que se tomó en cuenta proveer el mayor grado de soporte o flexibilidad a un usuario o administrador que presente dicha necesidad.

Finalmente, como trabajos futuros, se propone portar o migrar este demonio a otros sistemas operativos basados en Unix, como los sistemas BSD, Solaris, AIX, etc. Si bien gran parte del código implementado en este trabajo especial de grado es portable entre sistemas derivados de Unix, existen algunas características dependientes del sistema operativo que necesitan ser reescritas. También se propone agregar autenticación entre nodos, con alguno de los posibles métodos que sugiere el RFC 4795 [**3**].

Bibliografía

[1] P. Mockapetris, *Domain Names - Concepts And Facilities*.: RFC 1034, Noviembre 1987.

[2] P. Mockapetris, *Domain Names - Implementations and Specification*.: RFC 1035, Noviembre 1987.

[3] B. Aboba, D. Thaler, and L. Esibov, *Link-Local Multicast Name Resolution (LLMNR)*.: RFC 4795, Enero 2007.

[4] Microsoft, *MS-LLMNRP: Link Local Multicast Name Resolution (LLMNR) Profile*.: Microsoft, Enero 2013.

[5] S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*.: RFC 2460, Diciembre 1998.

[6] J. Davies, *Understanding IPv6*.: Microsoft Press, 2012.

[7] R. Hinden and S. Deering, *IP Version 6 Addressing Architecture*.: RFC 4291, Febrero 2006.

[8] C. Huitema and B. Carpenter, *Deprecating Site Local Addresses*.: RFC 3879, Septiembre 2004.

[9] R. Hinden and S. Deering, *IPv6 Multicast Address Assignments*.: RFC 2375, Julio 1998.

[10] R. Droms et al., *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*.: RFC 3315, Julio 2003.

[11] O. Troan and R. Droms, *IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6*.: RFC 3633, Diciembre 2003.

[12] Javvin Technologies, *Network Protocols Handbook*.: Javvin Technologies, 2005.

[13] S. Cheshire and M. Krochmal, *Multicast DNS*.: RFC 6762, Febrero 2013.

[14] K. Beck and C. Andres, *Extreme Programming Explained*.: Addison-Wesley, Noviembre 2004.

[15] A. Cockburn, *Agile Software Development*.: Addison-Wesley, Octubre 2001.

[16] D. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*.: O'Reilly Media, Diciembre 2005.

[17] R. Braden, *Requirement for Internet Hosts -- Aplication and Support*.: RFC 1123, Octubre 1989.

[18] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov, *Linux Netlink as an IP Services Protocol*.: RFC 3549, Julio 2003.

[19] M. Andrews, *Negative Caching of DNS Queries*.: RFC 2308, Marzo 1998.

[20] C. Benvenuti, *Understanding Linux Networking Internals*.: O'Reilly Media, Enero 2005.

[21] K. King, *C Programming: A Modern Approach*.: W. W. Norton & Company, Abril 2008.

[22] S. Loosemore, R. Stallman, R. McGrath, A. Oram, and U. Drepper, *GNU C Library Reference Manual*.: Free Software Foundation.

[23] R. Stevens, *Unix Network Programming*.: Prentice Hall PTR, Noviembre 2003.

[24] K. Fall and R. Stevens, *TCP/IP Illustrated: Volume 1*.: Addison-Wesley, Diciembre 2011.