

**INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY**

Campus Monterrey

Escuela de Ingeniería y Tecnologías de Información
Programa de Graduados



**TECNOLÓGICO
DE MONTERREY®**

PROPUESTA DE METODOLOGÍA DE BENCHMARKING PARA APLICACIONES
NAT64

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER
EL GRADO ACADÉMICO DE:
MAESTRIA EN CIENCIAS EN TECNOLOGÍA INFORMÁTICA

POR:

MIGUEL ALEJANDRO GONZÁLEZ GONZÁLEZ

MONTERREY, N.L.

MAYO DE 2013

**INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES
DE MONTERREY**
ESCUELA DE INGENIERÍA Y TECNOLOGÍAS DE INFORMACIÓN
PROGRAMA DE GRADUADOS

Los miembros del Comité de Tesis recomendamos que la presente tesis de Miguel Alejandro González González sea aceptada como requisito parcial para obtener el grado académico de Maestría en Ciencias en Tecnología Informática.

Comité de Tesis:

Dr. Juan Arturo Nolazco Flores

Asesor de Tesis

MSc. Alejandro Parra Briones

Sinodal

MSc. Roberto Aceves

Sinodal

Dra. Lorena Guadalupe Gómez Martínez

Directora de la Maestría en Ciencias en Tecnología Informática

Mayo de 2013

**PROPUESTA DE METODOLOGÍA DE BENCHMARKING PARA
APLICACIONES NAT64**

POR:

MIGUEL ALEJANDRO GONZÁLEZ GONZÁLEZ

TESIS

MAESTRÍA EN CIENCIAS EN TECNOLOGÍA INFORMÁTICA

Presentada al Programa de Graduados de la Escuela de Ingeniería y Tecnologías de
Información

Este trabajo es requisito parcial para obtener el grado de Maestría en Ciencias en
Tecnología Informática

**INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES
DE MONTERREY**

MAYO DE 2013

Resumen

Esta tesis de maestría presenta una propuesta de una metodología para realizar el benchmarking a aplicaciones NAT64, un mecanismo de transición IPv4-IPv6 que traduce los encabezados de los paquetes de red. Para proponer la metodología, se realizó una investigación previa sobre evaluaciones y pruebas que se hayan hecho con NAT64, por otro lado se analizó a detalle cada implementación de código abierto que fue encontrada hasta la fecha actual de la investigación.

El benchmarking consiste en dos fases: una etapa de funcionalidad y otra desempeño. La etapa de funcionalidad prueba la conectividad en los sistemas y la etapa de desempeño mide éste con la ayuda de métricas como *throughput*, latencia, utilización de CPU y de memoria. Dentro de los sistemas NAT64 probados se encuentran: Ecdysis, linuxnat64 de Lithuania, TAYGA, NAT64 de OpenBSD 5.1 y la implementación de NAT64 del ITESM-NIC.mx

Este benchmarking tiene como propósito ayudar a los desarrolladores de NAT64 dentro del ITESM a probar la aplicación y compararla con las que están actualmente en el mercado, también puede ser usada por administradores de red que planean que hacer un despliegue de NAT64 dentro de su empresa y de esta manera encontrar cuál de las aplicaciones disponibles se adecua a sus necesidades.

Reconocimientos

La realización de esta investigación de tesis de maestría fue posible, en primer lugar, a la orientación y ayuda económica brindada por Dr. Juan Arturo Nolazco Flores, director del departamento de Ciencias Computacionales del Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey.

Como es de entender, se agradece al Comité de Tesis, particularmente a Roberto Aceves por su dirección y ayuda constante, en especial por su orientación metodológica y por su continuo estímulo durante todo el proceso hasta al final del mismo.

Se agradece a todas aquellas personas que en forma directa o indirecta contribuyeron a que este trabajo de investigación pudiera llevarse a cabo. Por último un agradecimiento, a mi madre por su constante paciencia y apoyo que siempre ha demostrado.

Índice

Capítulo 1 Planteamientos generales	9
Antecedentes	9
Planteamiento del problema.....	11
Definición de variables.....	12
Objetivos.....	12
<i>Objetivo general.....</i>	12
<i>Objetivos específicos.....</i>	12
Justificación	13
Contexto de la investigación	13
Alcances y limitaciones del estudio	14
Capítulo 2 Marco Teórico	15
NAT64, estándar de la IETF.....	15
Propuestas existentes de NAT64	17
<i>Ecdysis.....</i>	17
<i>linuxnat64</i>	18
<i>nat64.....</i>	19
<i>OpenBSD NAT64</i>	20
<i>TAYGA.....</i>	20
<i>NAT64 ITESM-NIC.mx</i>	21
Metodologías de evaluación y pruebas	22
<i>Estudio 1: Mecanismos de transición IPv4-IPv6.....</i>	22
<i>Estudio 2: Compatibilidad de aplicaciones.....</i>	23
<i>Estudio 3: NAT64 vs. NAT44.....</i>	24
<i>Estudio 4: Pruebas con protocolos.....</i>	25
<i>Estudio 5: Análisis de paquetes / uso cotidiano.....</i>	25
<i>Estudio 6: Falla en flujos.....</i>	26
<i>Estudio 7: Análisis de sobrecarga</i>	26
Análisis de Estructuras de Datos utilizadas	27
<i>Slab Allocator.....</i>	27
<i>Hash list</i>	27
<i>Red-Black Tree</i>	28
<i>Hash table.....</i>	28
<i>Análisis de estructuras de datos</i>	29
Capítulo 3 Definición de la propuesta de Benchmarking para aplicaciones NAT64.....	31
Etapa de funcionalidad.....	32
Etapa de desempeño	33
Experimento.....	37
<i>Prueba de concepto</i>	37
Capítulo 4 Resultados	38
Resultados de la etapa de funcionalidad	38
Resultados de la etapa de desempeño	39
<i>Pruebas del protocolo TCP</i>	39
<i>Pruebas del protocolo UDP</i>	42
<i>Pruebas del protocolo ICMP</i>	45

<i>Observaciones</i>	48
<i>Análisis de la prueba de concepto</i>	49
<i>Análisis de resultados</i>	49
Capítulo 5 Conclusión....	51
Trabajo futuro	52
Anexos.....	53
I. Levantar ambiente de pruebas (PASO 1)	53
1) <i>Topología del ambiente de pruebas</i>	54
2) <i>Configuración de red de nodo cliente IPv6</i>	54
3) <i>Configuración de red de nodo cliente IPv4</i>	55
4) <i>Configuración de red de nodo servidor Linux</i>	55
5) <i>Configuración de red de nodo servidor OpenBSD</i>	56
II. Configuración de NAT64 (PASO 2).....	56
1) <i>Descarga de kernels de Linux probados</i>	56
2) <i>Configuración de Ecdysis</i>	57
3) <i>Configuración linuxnat64 de Lithuania</i>	59
4) <i>Configuración de TAYGA</i>	61
5) <i>Configuración de NAT64 en OpenBSD</i>	63
6) <i>Configuración de ITESM-NIC-2</i>	63
7) <i>Configuración de ITESM-NIC</i>	64
III. Configuración de equipo de red (PASO 3)	65
a. <i>Configuración de enrutador Cisco IPv4</i>	66
b. <i>Configuración de enrutador Cisco IPv6</i>	67
IV. Configuración de software para pruebas (PASO 4)	69
1) <i>Configuración para NTP</i>	69
2) <i>Instalación de traceroute</i>	69
3) <i>Instalación de SSH</i>	70
4) <i>Instalación de Apache2</i>	70
5) <i>Instalación de Apache Bench</i>	70
6) <i>Configuración de servidor DNS</i>	70
7) <i>Configuración de servidor DNSSEC</i>	72
8) <i>Instalación de DNSperf</i>	75
9) <i>Instalación de Observium</i>	75
V. Ejecución del benchmarking (PASO 5).....	78
1) <i>Comandos para la prueba de funcionalidad</i>	78
2) <i>Comandos para la prueba de desempeño</i>	78
3) <i>Ejemplo de resultados de confiabilidad</i>	79
Bibliografía	81

Índice de Figuras

<i>Figura 1 Ambiente de pruebas.....</i>	32
<i>Figura 2 Modelo de benchmarking para aplicaciones NAT64.....</i>	36
<i>Figura 3 Resultados TCP de paquetes pequeños.....</i>	41
<i>Figura 4 Resultados TCP de paquetes grandes</i>	42
<i>Figura 5 Resultados UDP paquetes pequeños.....</i>	44
<i>Figura 6 Resultados UDP paquetes grandes.....</i>	45
<i>Figura 7 Resultados ICMP paquetes pequeños.....</i>	46
<i>Figura 8 Resultados ICMP paquetes grandes.....</i>	47
<i>Figura 9 Uso de CPU.....</i>	48
<i>Figura 10 Uso de memoria física.....</i>	49
<i>Figura 11 Topología del ambiente de pruebas.....</i>	54
<i>Figura 12 Instalación de Ecdysis.....</i>	59
<i>Figura 13 Instalación de linuxnat64.....</i>	60
<i>Figura 14 Instalación de TAYGA.....</i>	62
<i>Figura 15 Instalación de ITESM-NIC-2.....</i>	64
<i>Figura 16 Instalación de ITESM-NIC.....</i>	65

Índice de Tablas

<i>Tabla 1 Diferencias entre NAT64 con estados y sin estados.....</i>	16
<i>Tabla 2 Complejidad en el caso promedio.....</i>	29
<i>Tabla 3 Complejidad en el peor caso.....</i>	29
<i>Tabla 4 Complejidad de hash list.....</i>	29
<i>Tabla 5 Aplicaciones a probar.....</i>	31
<i>Tabla 6 Hardware para clientes.....</i>	33
<i>Tabla 7 Hardware para servidores</i>	33
<i>Tabla 8 Herramientas para pruebas.....</i>	35
<i>Tabla 9 Resultados de funcionalidad.....</i>	38
<i>Tabla 10 Prueba de desempeño para TCP.....</i>	39
<i>Tabla 11 Prueba de desempeño para UDP.....</i>	43
<i>Tabla 12 Prueba de desempeño para ICMP.....</i>	45
<i>Tabla 13 Resultados de benchmarking</i>	50
<i>Tabla 14 Resultados de t-student de paquetes pequeños TCP.....</i>	79
<i>Tabla 15 Resultados de t-student de paquetes grandes TCP</i>	80

Índice de Ecuaciones

<i>Ecuación 1 Fórmula para calcular el rendimiento de HTTP</i>	34
<i>Ecuación 2 Fórmula para calcular el RTT promedio de ICMP.....</i>	34
<i>Ecuación 3 Fórmula para calcular la latencia promedio en DNS.....</i>	35
<i>Ecuación 4 Fórmula para calcular el nivel de confianza.....</i>	40

Capítulo 1 Planteamientos generales

Antecedentes

En los últimos años, la demanda de direcciones IP ha crecido considerablemente y la distribución de éstas en la versión 4 está cerca de alcanzar su límite. Debido a esto, diversos grupos han empezado a buscar métodos para traducir la tecnología de IP versión 4 (IPv4) a IP versión 6 (IPv6) por medio de una transición no súbita, ya que si la mayoría de los servicios de IPv4 se actualizaran a IPv6 esto podría traer un problema significativo a nivel mundial para quienes no tienen acceso a IPv6. IPv6 fue diseñado por la Internet Engineer Task Force (IETF) en 1995, la principal ventaja de esta tecnología es que puede proporcionar más que suficientes direcciones IP para el creciente número de dispositivos conectados a Internet. La versión anterior del protocolo de Internet, IPv4, no puede hacerlo. Este problema se remonta desde el diseño de TCP/IP en 1973 cuando Vinton Cerf y Robert Kahn, crearon la familia de protocolos usados actualmente en Internet. IPv4 usa un esquema de direcciones de 32 bits, esto significa que tiene un espacio de direcciones posibles de 2^{32} o alrededor de 4,300 millones.

Alrededor de los años noventa se creó el protocolo NAT (*Network Address Translation*), para remediar el problema del agotamiento de las direcciones IP, después que las personas se dieron cuenta que 4,300 millones no iban a ser suficientes para todo el mundo y algún día se acabarían. La tecnología NAT (usada en los enrutadores, servidores y *firewalls*) traduce un tipo de dirección de Internet a otro. Por ejemplo, dentro de las redes hoy en día, NAT traduce las direcciones IPv4 públicas en direcciones privadas, utilizadas dentro de una organización, a una dirección pública individual, utilizada en Internet. Lamentablemente, el agotamiento de las direcciones IP de la *Internet Assigned Numbers Authority* (IANA) sucedió en febrero de 2011 y todos los *Regional Internet Registry* (RIR) acabarán con las direcciones IP para 2014 (Dan, 2010).

Una de las ventajas principales de IPv6 es que cambia el esquema de direcciones por uno de 128 bits, con un espacio de direcciones posibles de 2^{128} , alrededor de 3.4×10^{38} dispositivos diferentes pueden conectarse a Internet (Geer, 2008). Pero una red IPv6 no se puede comunicar con una red IPv4 y viceversa, debido al cambio de esquema de direcciones y que los encabezados de los paquetes no están construidos de la misma manera. Para solucionar esto, existen mecanismos de transición de IPv4/IPv6.

Ahora muchas empresas e ISPs (*Internet Service Providers*) tratan de actualizarse a IPv6. Sin embargo, la transición de IPv4 a IPv6 no es una situación simple por si misma puesto que presenta el problema de que el equipo antiguo, que no cuente con una interfaz que de soporte a IPv6, no podrá conectarse a la red después de que se termine la transición. También, el actualizarse a IPv6 representa un gran costo para las empresas, ya que significa cambiar gran parte de su infraestructura de red,

seguramente adquirir nuevo equipo y en el peor de los casos, cambiar la aplicación que se esté usando porque no soporta IPv6.

Algunos mecanismos de transición de IPv4/IPv6, descritos en (ITESM, 2011a) se mencionan a continuación:

- 4in6. Encapsula a un paquete IPv4 dentro de un paquete de IPv6 para que éste sea el que lo transmita en la red.
- 6in4. Encapsula el tráfico IPv6 configurado en túneles de IPv4. Los puntos de fin se configuran estáticamente.
- 6over4. Transmite paquetes encapsulados en IPv6 utilizando IPv4 como una capa de enlace de datos virtual.
- 6to4. Encapsula los paquetes IPv6 en IPv4 y los decapsula si van a una interfaz IPv6.
- DS-Lite. Encapsula paquetes IPv4 en IPv6 y los decapsula para mandarlo por NAT.
- IVI. Es una técnica de traducción IPv4/IPv6 sin estados, permite mantener la transparencia entre dos extremos de una conexión.
- ISATAP. Técnica que permite transmitir paquetes IPv6 sobre una red IPv4, no requiere la infraestructura de IPv4.
- Túneles Teredo. Encapsula paquetes IPv4 en paquetes UDP y es reenviado por túneles Teredo.

Los mecanismos de transición se pueden generalizar en dos tipos: traducción de encabezados y encapsulamiento mediante túneles. La IETF, eligió a la técnica de traducción de encabezados para que se comunique al mundo IPv4 con el mundo IPv6. Durante un tiempo la IETF se dedicó a estudiar cuatro diferentes tecnologías o propuestas para se convierta en la tecnología estándar para usar en este caso (Geer, 2008). Las tecnologías son:

- DS-Lite. Como ya se mencionó, encapsula paquetes IPv4 dentro de los datos de un paquete de IPv6 y se mandan a través de un túnel hasta llegar a un NAT. Al decapsular el paquete de IPv6 se encontrará el paquete de IPv4. Esta tecnología combina traducción con túneles.
- IVI. Protocolo desarrollado en China, es un mecanismo de mapeo y traducción implementado en un Gateway que conecta IPv4 con IPv6.
- NAT64. Los investigadores de la Universidad de España Carlos III de Madrid y el Instituto Madrileño de Estudios Avanzados desarrollaron esta propuesta. La tecnología permite que los dos protocolos de Internet puedan trabajar juntos mediante la traducción de los encabezados de IPv6 y de IPv4 usando el algoritmo *Stateless IP/ICMP* (Internet Control Message Protocol) para la traducción. Con SIIT, un enrutador interpreta un encabezado IPv4 y crea un encabezado de IPv6 paralelo, y viceversa. NAT64 trabaja en conjunto con DNS64, un mecanismo para permitir que los registros que se asocian las direcciones IPv6 se comunique con los registros de nombre de dominio que se asocian las direcciones IPv4 con un nombre de dominio específico.

- NAT6. Esta tecnología, hecha por Cisco, traduce únicamente de IPv6 a IPv4 y no al revés. Esta propuesta requiere que una aplicación en una máquina IPv6, añada un prefijo IPv6 a una dirección IPv4 para crear una dirección IPv6.

Sin embargo, como se sabe, NAT rompe la conectividad rápida, transparente, directa y de extremo a extremo que suelen utilizar las aplicaciones de tiempo real, como por ejemplo, la transmisión de video. Por otro lado, la encapsulación de paquetes y uso de túneles añaden demasiada complejidad a la red, lo cual la hace difícil de escalar e incrementar el nivel de seguridad.

Se ha prestado mucha atención a la situación que le permite a clientes IPv6, el acceso a servidores IPv4 a través de Internet mediante dos mecanismos: traducción IPv6/IPv4 sin estados (*stateless*) y con estados (*stateful*) (Dan, 2010). Tanto la propuesta de IVI como la propuesta NAT64 han llamado la atención, ya que NAT64 implementa una traducción con estados e IVI implementa una traducción sin estados. Existe, y seguirá existiendo al menos por un tiempo, contenido considerable sólo de IPv4 en Internet. Por lo tanto, para proporcionar a su creciente base de suscriptores con acceso a Internet IPv4, los ISP deben comenzar la agregación de varios suscriptores en una sola dirección IPv4. Este tipo de comunicación se logra con NAT64.

Actualmente existen productos comerciales que realizan alguno de los tipos de traducción de NAT64. Empresas como Cisco, Microsoft, Juniper ya venden productos tanto de hardware o de software para solucionar el problema de la comunicación entre IPv6 e IPv4, basándose en los estándares de la IETF. Productos de software libre también han surgido, en especial para sistemas operativos como Linux y OpenBSD. Pero los productos para la plataforma de Linux no han sido aceptados por la comunidad, no han sido actualizados en mucho tiempo y no se tiene mucha documentación sobre éstos proyectos. Se podría decir que no todos se apegan al estándar de la IETF, el RFC 6146, y les faltan funcionalidades por implementar.

El Tecnológico de Monterrey y la empresa NIC México iniciaron un proyecto de investigación y desarrollo en Agosto de 2011, que se enfocó a buscar una alternativa viable de transición de una red IPv4 a IPv6. Por lo que se inició el desarrollo de otra implementación de NAT64 con estados. Para que este proyecto llegue a su fin exitosamente es esencial realizar pruebas de desempeño y de estrés, es decir el producto final deberá pasar por un proceso de validación y verificación de calidad para comprobar su valor dentro del mercado de NAT64 y otros mecanismos de transición.

Planteamiento del problema

El problema a resolver dentro de este documento consiste en establecer el proceso de pruebas o “benchmarking” por el cuál el producto deberá pasar, realizar éstas pruebas y comparar el desempeño contra otras implementaciones de NAT64 que ya se encuentren disponibles. Los resultados de este benchmarking de desempeño darán la pauta para saber si se tiene que arreglar algún problema con el producto que el Tecnológico de Monterrey se encuentra desarrollando, o para finalmente publicar dichos resultados explicando las ventajas de esta nueva implementación.

De acuerdo a lo anterior se plantea la siguiente pregunta de investigación:

¿Existe un modelo estándar para realizar un benchmarking de desempeño para el mecanismo de transición IPv4/IPv6 NAT64 con estados?

A continuación se observarán las variables que integran al problema de investigación.

Definición de variables

El problema de investigación está integrado de una variable independiente y una variable dependiente. Dichas variables establecen una relación funcional, es decir, revisar si existe un modelo estándar para hacer un benchmarking de desempeño para el mecanismo de transición IPv4/IPv6 NAT64 con estados.

Variable dependiente: el mecanismo de transición IPv4/IPv6 NAT64 con estados.

Variable independiente: un modelo estándar para realizar un benchmarking de desempeño.

Una vez establecidas las variables y con base a la pregunta de investigación que se planteó anteriormente, los objetivos de la investigación son los siguientes.

Objetivos

Objetivo general

El objetivo principal de este estudio es determinar si existe un método formal o estándar para realizar un benchmarking de desempeño para NAT64.

Objetivos específicos

Si no hay un modelo estándar, tanto en la literatura académica como científica que se haya revisado, se quiere **obtener** las bases para proponer un modelo estándar de pruebas para realizar un benchmarking de desempeño.

Se pretende **ejecutar** dicho modelo de benchmarking con diferentes implementaciones de NAT64 y se documentarán los resultados obtenidos.

Se pretende también **estudiar** cada implementación de manera que se tenga una noción técnica de esta, de forma que los resultados de las pruebas sean fácil de interpretar y permitan tomar una decisión informada sobre qué característica se necesita modificar en el NAT64 desarrollado por el Tecnológico de Monterrey para mejorar su desempeño.

Como objetivo final se quiere **modificar** el NAT64 del Tecnológico de Monterrey y demostrar que el benchmarking pueda reflejar si el desempeño del sistema empeora o

mejora gracias al cambio realizado. Esto tiene como propósito que el benchmarking sirva de ayuda a los desarrolladores, de cualquier sistema de NAT64, y puedan detectar fácilmente la manera de mejorar su sistema.

Justificación

El agotamiento de las direcciones IPv4 en el mundo y el advenimiento de la transición a redes IPv6 vislumbran la necesidad de un mecanismo que permita la coexistencia de ambas versiones del protocolo de internet (IP). Tanto sus características como la variedad de sus implementaciones en el mercado, comerciales y de código abierto, proyectan a NAT64 como un mecanismo de transición confiable y válido.

Es por ello que surge la necesidad de un modelo de benchmarking que permita realizar una comparación de las diversas implementaciones de NAT64. Este modelo facilitará, por ejemplo, la toma de una decisión informada sobre cuál de éstos productos podría tener un mejor desempeño en una empresa.

Contexto de la investigación

El Tecnológico de Monterrey y NIC.mx iniciaron un proyecto para implementar un mecanismo de transición entre redes IPv4 e IPv6. Después realizar una investigación (ITESM, 2011a) y de analizar las opciones disponibles se optó por implementar NAT64.

El producto desarrollado por NIC y el Tecnológico es un sistema computacional para la implementación de *stateful* NAT64 (NAT64 en base a estados) en la que, en conjunto con un servidor de DNS64, se implementa la estrategia de transición de IPv4 a IPv6 de tipo *header translation* (traducción de encabezados) en donde solo se puede inicializar la conexión desde un cliente IPv6 a un servidor IPv4. Si bien existen otras estrategias como *dual-stack* y *tunneling*, esta implementación considera que, en base a lo descrito en el RFC6146, lo adecuado es mantener el diseño basado en estados con una traducción del encabezado de los paquetes de información recibidos durante la comunicación. La solución que se propone trabaja sobre el sistema operativo Linux como un módulo de *kernel*. (ITESM, 2011b)

El proyecto actualmente ha pasado por dos ciclos de desarrollo, en donde estudiantes de profesional han estado trabajando sobre el código, una vez que se terminen los últimos detalles faltantes, tendrá que pasar por una fase de pruebas de estrés y performance para determinar la calidad del producto. Dicha fase de pruebas se espera que pueda ser influenciada por este documento. La primera iteración del proyecto se enfocó en tener un producto funcional para los tres protocolos (UDP, TCP e ICMP), el código tiene muchas funciones heredadas de los proyectos: Ecdysis y linuxnat64 de Lithuania. Es esta versión la cual fue tomada para realizar las pruebas. Por otro lado, la segunda iteración del proyecto se enfocó en estar lo más apegado posible al RFC6146 y reescribir la mayoría del código.

Alcances y limitaciones del estudio

En el RFC 6144 (Baker, Li, Bao, & Yin, 2011) detallan los posibles escenarios en los cuales se puede utilizar la traducción IPv6/IPv4 para permitir la comunicación entre dos redes IPv6 e IPv4. En ese documento se describen 8 escenarios de comunicación. Los siguientes tres cubren la funcionalidad de *stateful* NAT64 y sirven como base para las pruebas descritas en este documento. *Stateless* NAT64 funciona en los escenarios 1, 2, 5 y 6; los escenarios 7 y 8 no tienen un mecanismo capaz de traducir la comunicación. La comunicación considera los protocolos UDP, TCP, ICMP e ICMPv6.

- Escenario 1: Comunicación entre red IPv6 e Internet IPv4

La red es sólo IPv6 pero se requiere comunicar con el servidor IPv4 a través de Internet.

- Escenario 3: Comunicación entre Internet IPv6 y red IPv4

Para sistemas antiguos que se encuentran en una red IPv4 y requieren todavía dar servicio a clientes IPv6 a través de Internet.

- Escenario 5: Comunicación entre red IPv6 y red IPv4

Si ambas redes existen en la misma organización y se requiere la comunicación entre ellas. Por ejemplo, se tienen servicios dentro de la organización que están localizados en servidores que se encuentran en la red IPv4 y los clientes están en la red IPv4.

De los tres escenarios de NAT64 *stateful*, solamente se probará el escenario número 5 de Comunicación entre red IPv6 y red IPv4, considerando que si se realiza la comunicación bajo este escenario se espera que funcione para los demás. Esta limitación se debe a que no tiene la infraestructura IPv6 necesaria dentro del Campus para abarcar los demás escenarios. La comparación contra otros mecanismos de transición queda fuera del alcance de este estudio. Más adelante se describirá en detalle el ambiente de pruebas que se utilizó para realizar este estudio.

En el siguiente capítulo se presenta el Marco Teórico, en donde se muestra la investigación realizada y el estado del arte que ayudan a formar la base para la propuesta del benchmarking. El capítulo 3 describe la propuesta de benchmarking para aplicaciones NAT64 a detalle. El capítulo 4 muestra los resultados obtenidos del benchmarking. Finalmente el capítulo 5 contiene las conclusiones de la tesis.

Capítulo 2 Marco Teórico

En este capítulo se presentan los hallazgos sobre el tipo de pruebas que se realizan a NAT64 actualmente. De la misma manera se presenta información sobre las funcionalidades que provee el estándar NAT64 y sobre varias de las implementaciones de código abierto que actualmente se encuentran en el mercado. Al final se presenta un análisis de estructuras de datos que son usadas en dichas implementaciones.

NAT64, estándar de la IETF

NAT64 es un mecanismo de transición basado en estados que permite dar conectividad IPv4 a clientes que son solamente IPv6. Se define en el RFC6146 (Bagnulo, Matthews, & van Beijnum, 2011) y abarca los protocolos UDP, TCP e ICMP.

Un mecanismo basado en estados mantiene en memoria información de la conexión, es decir estados, por cada paquete traducido. El concepto de estados proviene de *stateful firewalls*, que mantienen también en memoria estados sobre conexiones como TCP y UDP. Sólo los paquetes que coincidan con una conexión activa serán permitidos por el *firewall*, mientras que los otros serán rechazados.

NAT64 es un mecanismo que mantiene una relación 1-N, esto significa que muchos clientes IPv6 tienen que compartir una misma dirección IP de IPv4. En teoría, NAT64 sólo necesita n direcciones IPv4, n debe ser una cantidad razonable, para representar a clientes IPv6. Por otro lado, NAT64 no necesita de una subred especial dedicada para representar el espacio de direcciones IPv4. Todas las direcciones IPv4 son embebidas con el prefijo especial 64:ff9b::/96 para que pueda funcionar la traducción.

Para mantener la información sobre las conexiones, NAT64 usa dos tablas por cada protocolo, la *Binding Information Base* (BIB) y la tabla de Sesión. La tabla de sesión contiene la siguiente información:

- Dirección IPv6 origen y puerto origen IPv6 (ID en el caso de ICMP). La dirección del cliente de la red IPv6.
- Dirección IPv6 destino y puerto destino IPv6 (ID en el caso de ICMP). La dirección del prefijo del servidor NAT64 embebida con la dirección IPv4.
- Dirección origen IPv4 y puerto origen IPv4 (ID en el caso de ICMP). La dirección otorgada por la pool de direcciones IPv4.
- Dirección destino IPv4 y puerto destino IPv4 (ID en el caso de ICMP). La dirección destino de la red IPv4.
- Tiempo de vida: El tiempo de vida restante de la conexión.

La tabla de BIB contiene las siguientes direcciones de transporte:

- Dirección origen IPv6 y puerto origen IPv6 (ID en el caso de ICMP). La dirección del cliente de la red IPv6.
- Dirección origen IPv4 y puerto origen IPv4 (ID en el caso de ICMP). La dirección otorgada por la pool de direcciones IPv4.

El estándar de NAT64 propone el uso de la funcionalidad *Endpoint Independent Mapping*, con el fin de usar el mismo mapeo para todas las sesiones. Esto implica a una determinada dirección de transporte IPv6 de un cliente IPv6, independientemente de la dirección de transporte del cliente IPv4 involucrada en la comunicación.

El filtrado dentro de NAT64 se puede clasificar en dos tipos:

- *Endpoint Independent filtering*:
NAT64 sólo filtra paquetes IPv4 destinados a una dirección de transporte para el que no hay un estado en el NAT64, independientemente de la dirección de origen del transporte IPv4. El NAT reenvía los paquetes destinados a cualquier dirección de transporte para el que tiene un estado.
- *Address Dependent filtering*:
NAT64 sólo filtra paquetes entrantes IPv4 destinados a una dirección de transporte para el que no hay un estado. Además, el NAT64 filtrará los paquetes IPv4 entrantes procedentes de una dirección IPv4 X, con destino a una dirección de transporte para el que tiene un estado, sí el NAT64 no ha enviado paquetes a X antes independientemente del puerto usado por X.

De acuerdo a lo ya mencionado, a continuación se presenta una comparación de NAT64 con estados y sin estados. Cabe mencionar que es necesario tomar en cuenta que los dos mecanismos realizan la tarea de traducir los paquetes IPv4 en paquetes IPv6 y viceversa, sin embargo sí hay diferencias importantes. La tabla 1 proporciona una descripción general de las diferencias más relevantes.

NAT64 sin estados	NAT64 con estados
Traducción 1-1	Traducción 1-N
No conserva direcciones IPv4	Conserva direcciones IPv4
Asegura la escalabilidad y la transparencia de direcciones <i>end-to-end</i>	Usa una pool de direcciones de transporte IPv4, por lo que rompe el principio <i>end-to-end</i>
Ningún estado o enlace es creado en la traducción	Se crea un estado y un enlace por cada traducción única
Se requieren direcciones IPv6 que puedan ser traducidas a IPv4.	No existe ningún requisito sobre la naturaleza de asignación de dirección IPv6

Tabla 1 Diferencias entre NAT64 con estados y sin estados

Propuestas existentes de NAT64

Dentro de esta sección se describen a detalle algunas implementaciones de NAT64 de código abierto. Se podría decir que éstas son las más relevantes que se encuentran en Internet hasta el momento, pero no son todas las disponibles. Algunos criterios para elegir éstas aplicaciones fueron: la influencia que han causado, disponibilidad, nivel de documentación.

Ecdysis

El proyecto Ecdysis fue iniciado por la empresa Viagénie (Dionne, Perreault, & Blanchet, 2010), con el objetivo de desarrollar una implementación de código abierto de un Gateway que pudiera traducir direcciones IPv6/IPv4. Se inició durante las etapas cuando los estándares de la IETF se encontraban en etapa de “borrador”. El esfuerzo del equipo Ecdysis produjo dos módulos diferentes uno para NAT64 y uno para DNS64 para dos sistemas operativos, Linux y OpenBSD. El modulo de DNS64 fue implementado como un parche a dos servidores DNS populares BIND y Unbound. Por otro lado, el módulo de NAT64 fue desarrollado modificando *pf*, la herramienta de OpenBSD que se encarga implementar un firewall y también NAT44 o NAT de IPv4. En el caso de Linux se utilizó *Netfilter*, que de igual manera es el código para implantar un *firewall* o un servidor NAT.

Las dos implementaciones son muy diferentes. La versión de Linux utiliza un *hook* de Netfilter para capturar paquetes de entrada y después procesarlos. El término *hook* cubre una gama de técnicas utilizadas para modificar o aumentar el comportamiento de un sistema operativo, de las aplicaciones, o de otros componentes de software por llamadas de función de interceptación o mensajes o eventos intercambiados entre componentes de software. El código que se encarga de este tipo de llamadas a funciones interceptadas, eventos o mensajes se le llama un "*hook*". Para guardar las sesiones, se utiliza un árbol como estructura de datos, específicamente un *Red-Black tree* utilizando la implementación que se encuentra dentro del kernel de Linux `<linux/rbtree.h>`. Finalmente hace uso de una interfaz virtual por donde los paquetes traducidos son mandados con la función `netif_rx()`. La implementación de Linux es en su mayoría independiente del código del seguimiento de conexión existente dentro de Netfilter, llamado *conntrack*.

Por otro lado, la implementación de OpenBSD está muy fuertemente integrada con *pf*, esto le hace beneficiarse automáticamente de compatibilidad con otras partes del sistema. Por ejemplo, la configuración se guarda en el archivo `pf.conf`, `pfsync` (protocolo para intercambiar los estados entre firewalls) intercambia estados de NAT64 con diferentes servidores *pf*, e integración con el protocolo CARP (*Common Address Redundancy Protocol*) que sirve para proveer redundancia. Los paquetes son traducidos en la entrada de *pf* y el paquete traducido es mandado a la función `ipv4_input()` ó `ip6_input()` dependiendo de que se quiere traducir, como si hubiese llegado directamente a por esa entrada. El paquete original se deja de procesar, reemplazándolo con un apuntador a nulo.

Las sesiones son guardadas, aprovechando la manera en la cual *pf* guarda sus conexiones con estados en el NAT tradicional. En este caso se guarda un llave al inicio del procesamiento y una al final para identificar respectivamente a IPv4 e IPv6. Es por la integración con *pf*, que la implementación no cumple del todo con el estándar de la IETF. Por ejemplo en lugar de permitir un mapeo independiente en los extremos, un mapeo dependiente en dirección y puerto es utilizado para tener un manejo más eficiente de las direcciones IPv4.

[linuxnat64](#)

Dentro de la universidad de Kaunas, Lituania se desarrolló *linuxnat64*, otra implementación de NAT64 para Linux. Este proyecto fue presentado en la conferencia *TERENA Networking* en 2010, en la que se desplegó el servicio entre la red LITNET y la red de la universidad de Kunas. Desde entonces el servicio no ha sido apagado y esta disponible para el público. Hasta el momento se ha utilizado principalmente para realizar experimentos con sólo clientes IPv6 y acceder a redes IPv4. Para empezar a utilizar este servicio, sólo hay que configurar la dirección correcta del servidor DNS64.

De acuerdo a (Kriukas, 2010), *linuxnat64* es un módulo para el kernel de Linux basado en la etapa de borrador del RFC 6146, sin embargo se trató de independizar lo más posible del kernel para evitar conflictos con las diferentes versiones de éste. Se escogió implementarlo como un módulo, principalmente por razones de facilidad en las pruebas. Otra opción que los desarrolladores tomaron en cuenta fue implementarlo como un parche al código del *kernel*, pero esto requeriría compilar el *kernel* cada vez que se deseara probar la aplicación. Para configurar este módulo se utiliza un mecanismo de comunicación que intercambia parámetros entre Userspace y Kernelspace, debido a que no se requiere un gran número de elementos de configuración de parte del usuario, y a que ésta usualmente no cambia.

Hay que tomar en cuenta que esta implementación de NAT64 usa Netfilter en conjunto con una interfaz virtual desarrollada con un *netdevice* o interfaz virtual dentro del *kernel* de Linux. El desarrollador afirma que el uso de una interfaz virtual resuelve tres problemas en el envío de los paquetes, que se detallan a continuación:

- Si no se usa una interfaz virtual se asegura la destrucción de la integridad del modelo de ruteo en Linux.
- Se limita la posibilidad de administrar el flujo de los paquetes por medio de reglas para un firewall. Un paquete IPv6 pasa por las cadenas de Netfilter: PREROUTING, OUTPUT y POSTROUTING. Al no usar una interfaz virtual, las cadenas se mezclan con paquetes de flujo IPv4, esto también sucede en las cadenas de IPv4, que se mezclan con flujo de IPv6.
- Se necesita implementar un módulo para que haga la decisión de ruteo.

Al usar una interfaz virtual, un paquete IPv6 pasa por PREROUTING, OUTPUT y POSTROUTING, después llega a la interfaz virtual, donde es traducido y mandado a la cadena OUTPUT de IPv4. Los paquetes que regresan de IPv4 son tomados de la cadena INPUT, y después son traducidos y mandados a la interfaz virtual. Este proceso simplifica la transmisión de paquetes y la ventaja es que el flujo de paquetes IPv6 es generado por la interfaz virtual como si fuera una interfaz cualquiera. Sin embargo, no

cambia tanto en el lado IPv4. Dentro de la interfaz virtual los paquetes son mandados con la función `netif_rx()`.

El módulo linuxnat64 usa como estructura de datos una lista, donde guarda los registros de la BIB y la Sesión. Por ello el tiempo de búsqueda es del orden $O(n)$. Una entrada de la BIB de IPv6 o IPv4 (dirección de transporte) es identificada únicamente por un hash, en lugar de un par de direcciones de transporte. Debido a la manera en la cual los puertos son usados en la comunicación TCP/IP, las computadoras usan un puerto aleatorio para indicar la comunicación, por esto el autor afirma que una BIB tendrá usualmente sólo una sesión asociada y tendrá un tiempo de búsqueda $O(1)$, pero si tiene más el tiempo será $O(n)$.

Las pruebas realizadas en este trabajo fueron hechas usando la herramienta `tcpdump` que permite analizar el contenido de los paquetes. Se compararon a nivel hexadecimal los contenidos de los paquetes antes y después de ser traducidos de los tres protocolos (UDP, TCP e ICMP). También se usó la herramienta `tracepath` con el fin de pruebas.

[nat64](#)

El proyecto nat64 desarrollado en la Universidad de Liège (Rondou, 2011) es otra implementación de NAT64 de código libre. También es un módulo para *el kernel* de Linux, con la diferencia que esta desarrollado dentro del *framework* de Xtables. Una de las ventajas de este enfoque es que el *framework* trabaja fuera del *kernel* y no necesita que la computadora reinicie para que se pueda compilar o cargar el módulo. Xtables no depende de *un kernel* específico para funcionar. Otra diferencia es que ésta implementación tiene código tanto de Userspace como Kernelspace.

Es importante tomar en cuenta, ésta implementación maneja paquetes fragmentados. Según el autor, los fragmentos pueden ser manejados de dos maneras:

- Fragmento por fragmento De esta manera, los fragmentos se tratan uno por uno, a medida que llegan. Esta es la forma más eficaz de acuerdo al autor, de tratar con fragmentos, ya que no ocupa espacio ni tiempo para la reconstrucción. La aplicación de nat64 implementa el manejo de fragmentos de esta manera, en la comunicación de IPv6 a IPv4.
- Ensamblando el paquete. Los fragmentos se guardan hasta que el paquete completo se pueda reconstruir; el autor menciona que este proceso no es realmente eficaz. La ventaja de este enfoque es que servidor se puede asegurar que el paquete esté completo cuando salga por la interfaz y al mismo tiempo limitar el tamaño del MTU. Este método se utiliza en la comunicación de IPv4 a IPv6 y depende del módulo de kernel `nf_conntrack_defrag`.

Este proyecto hace uso de una serie de estructuras de datos para implementar la tabla de BIB y la tabla de Sesión. El autor comenta que “la elección de las estructuras de datos es esencial en el rendimiento del módulo. Aquí, decidí que prefiero el tiempo de ejecución sobre la ocupación de memoria, porque NAT64 será más probablemente utilizado en un ambiente tipo ISP, con una gran cantidad de memoria disponible.” (Rondou, 2011).

Dentro de las estructuras de datos que usa se encuentran la lista encadenada, una tabla de hashing y una cola priorizada que implementa el autor debido que el kernel no las ofrece de manera genérica o en el caso de la lista encadenada “no es usable”. También incluye dos estructuras para el manejo de la sesiones. El SessionMap que es una tabla de hashing con una lista encadenada como bucket. Y el SessionInfo que guarda información de la sesión y se almacena en el bucket.

La lista encadenada también es usada para mantener dos listas para TCP y UDP, una en donde se almacenen puertos menores a 1024 y la otra en donde estén los puertos de 1024 a 65535. La cola priorizada se usa para eliminar las sesiones expiradas. Se mantienen dos SessionMap junto con su cola priorizada por cada protocolo para guardar la sesión, cada una guarda información de IPv6 e IPv4 respectivamente.

OpenBSD NAT64

El sistema operativo OpenBSD a partir de la versión 5.1 incluye con una implementación de NAT64 dentro de *pf*, la herramienta para filtrado de paquetes que sirve como firewall. El desarrollo de esta implementación inició cuando los desarrolladores de Ecdysis mandaron un parche a lista de correos de OpenBSD con la propuesta de integrar su implementación de NAT64 al código fuente del sistema operativo. De acuerdo a los desarrolladores de OpenBSD, la versión que se liberó es una versión actualizada y reescrita en su mayoría, ya que la implementación de Ecdysis presentaba algunos inconvenientes: no estaba terminada, no estaba siendo actualizada, tenía una funcionalidad limitada y algunos errores.

El grupo Ecdysis menciona en (Dionne, et al., 2010) que no se pueden traducir las familias de direcciones de la misma manera que se traducen las direcciones en *nat-to* y *rdr-to*, debido a que la traducción de familias de direcciones no es compatible con el ruteo y requiere de reinyección de paquetes. Los desarrolladores de *pf* decidieron abordar este problema mediante la aplicación de reglas de NAT64 en la interfaz de entrada, tomando una decisión de ruteo e injectando el paquete a la interfaz de salida. Esto tiene la ventaja de no requerir que el paquete tenga que recorrer la cola de entrada ip/ip6.

El manejo de sesiones se implementó usando la idea original de Ecdysis, guardando llaves de estado en IPv6 e IPv4. Otro cambio notorio fue que se cambió la gramática para crear reglas de NAT64 en el archivo *pf.conf*.

TAYGA

TAYGA es una implementación de NAT64 sin estados que opera fuera del kernel de Linux. Utiliza el controlador TUN para el intercambio de paquetes IPv4 e IPv6 con el kernel. Su objetivo es proporcionar un servicio NAT64 de calidad a nivel producción para redes donde hardware dedicado sería un problema. De todas las implementaciones de NAT64 disponibles, TAYGA es la única que ha sido adoptada por la comunidad de código abierto a tal grado que se encuentra en los repositorios oficiales de distribuciones como Debian y Ubuntu.

Aunque TAYGA no es un NAT64 con estados, se puede configurar para que de una manera use estados. TAYGA carece la potencia y la flexibilidad disponible en iptables e implementaciones comerciales ya maduras, como NAT44. TAYGA convierte IPv6 a IPv4 de la manera más transparente posible, lo que permite a herramientas existentes que funcionan sólo con IPv4, ser utilizadas para manipular las sesiones que fluyen a través de TAYGA.

Si se necesita NAT64 con estados, usando esta implementación, se debe enrutar el camino IPv4 de TAYGA a través un NAT44 con estados. En teoría, herramientas como pf y iptables podrían servir para crear un NAT64 con estados usando TAYGA.

TAYGA se probó en el estudio (Llanto & Yu, 2012), del cuál se hablará más adelante.

NAT64 ITESM-NIC.mx

Este proyecto fue iniciado en agosto de 2011 por el Tecnológico de Monterrey y la empresa NIC México. De acuerdo a (ITESM, 2011b), la implementación de NAT64/DNS64 se desarrolló como un módulo del kernel de código abierto, estable y eficiente que corre sobre el sistema operativo Linux, en las versiones más recientes del kernel, siguiendo lo establecido en el RFC6146. Se desarrolló usando como base el kernel 2.6.38-10-server, la *Developer OpenSSL Library* (DNS64) y la *Developer IPtables Library* (NAT64).

Los principales componentes desarrollados son: a) determinar la tupla de entrada, b) filtrado y actualización de tablas, c) generar la tupla de salida y d) traducir el paquete, basadas en el proceso descrito en el RFC6146. Para la implementación, se creó una extensión “target” de Netfilter. Dicho “target” se colocó en el gancho PREROUTING de Netfilter y se le indicó recibir paquetes sin importar su protocolo, ya sea IPv6 o IPv4. El módulo utiliza *conntrack* de Netfilter para obtener la tupla de entrada. Es importante aclarar que esta implementación reutiliza código de Ecdysis y de linuxnat64. De Ecdysis se tomaron, por ejemplo, las funciones para calcular el checksum y para mandar paquetes a IPv6 y a IPv4. De linuxnat64 se tomaron las estructuras para el manejo de los registros de BIB y de Sesión.

Aunque este módulo todavía sigue en desarrollo, ya incluye elementos como rutas estáticas, manejo de hairpinning, una pool de IPv4, configuración mediante Netlink, y traduce los tres protocolos requeridos por el RFC6146: TCP, UDP e ICMP.

Se estableció que el módulo se usaría en la tabla “mangle” con el HOOK PREROUTING, puesto que un HOOK se usa para el manejo y manipulación de paquetes antes de que el sistema operativo decida qué hacer con ellos (ya sea que entren al sistema o se envíen a otro cliente). Por otro lado, se evitó el uso de interfaces externas y virtuales para mandar los paquetes, se utilizó código del kernel para conseguir una ruta y un dispositivo apropiado para mandar los paquetes, tanto de IPv6 como de IPv4. El módulo garantiza trabajar con diferentes versiones del kernel a partir de 2.6.38 en adelante, incluyendo a las versiones más nuevas como las 3.0.0. y 3.2.0.

Metodologías de evaluación y pruebas

En esta sección se describen una serie estudios relacionados con la evaluación de NAT64. Cada estudio utiliza una metodología diferente, pero el propósito de describir cada uno de éstos es obtener las bases para proponer una metodología para realizar un benchmarking de desempeño. También se busca obtener las métricas y técnicas de pruebas en común que tienen los estudio y consolidarlas en la propuesta que se describe en el siguiente capítulo.

Estudio 1: Mecanismos de transición IPv4-IPv6

En un estudio de la Universidad de Auckland (Yu & Carpenter, 2012), en donde se analiza el desempeño de algunos mecanismos de transición, como NAT-PT, NAT64 y un Proxy HTTP, se utilizó Ecdysis como implementación de un servidor NAT64. Las pruebas también se realizaron en una red IPv4 nativa y en una red IPv6 nativa, para comparar los resultados y observar si el proceso se hace más lento de lo normal. Cabe recalcar que en dicho estudio se eligió la versión de Linux para hacer las pruebas. Una de las pruebas consistió en enviar paquetes HTTP desde un cliente hacia un servidor pasando a través de los traductores y viceversa, con varios números de conexiones simultáneas, y se midió el RTT (Round Trip Time) de los paquetes devueltos.

Los paquetes tenían dos tamaños diferentes, identificados como "pequeño" y "grande". Un paquete pequeño se refiere a un paquete HTTP REQUEST de 107 bytes de tamaño, el paquete grande es un paquete HTTP REQUEST con un encabezado HTTP POST con 1200 bytes de caracteres aleatorios resultando en 1382 bytes de tamaño total. El cliente establece de 1 a 100 conexiones simultáneas con el servidor para cada serie de experimentos. Cada conexión se utiliza para enviar 10 mil paquetes HTTP REQUEST idénticos y recibir 10 mil HTTP RESPONSE cada vez. Como ambiente de pruebas, instalaron maquinas Linux con *kernel* 2.6.31-16 y usaron interfaces de red Gigabit Ethernet. El servidor Web tiene dos diferentes respuestas HTTP. Una de ellas es un pequeño HTTP RESPONSE que contiene un sólo *string* "a" y la otra es un gran HTTP RESPONSE con un *string* de caracteres aleatorios de 1083 bytes.

Los resultados se presentan en microsegundos, y reportan la mediana del RTT. Para este estudio en particular, NAT64 de Ecdysis, mostró un mejor rendimiento en comparación con los demás mecanismos de transición analizados. Diferentes pruebas fueron realizadas, una con pequeñas HTTP REQUEST tanto como HTTP RESPONSES y otra con pequeñas HTTP REQUEST en conjunto con grandes HTTP RESPONSES. Sin embargo NAT64 resulta, alrededor 45% más lento que las conexiones en una red IPv6 común y ese porcentaje incrementa cuando se incluyen más conexiones simultáneas. Es interesante como en la prueba de paquetes grandes, es decir con paquetes HTTP REQUEST y HTTP RESPONSE grandes y con un pequeño número de conexiones simultáneas, la mediana del RTT de NAT64 es la más grande entre los tres traductores comparados y 50% más lento que la conexión nativa con IPv6. Los investigadores deducen que no es **problema teóricamente** con NAT64, sino que podría ser un problema de implementación.

Respondiendo a éste estudio, los desarrolladores de Ecdysis realizaron pruebas y, aunque dicen no poder reproducir los resultados sus pruebas internas demuestran que la implementación de OpenBSD tiene mucho mejor desempeño que la implementación de Linux. El grupo que desarrolló Ecdysis mostró sus pruebas que también incluyen paquetes grandes y pequeños. La metodología de éstas pruebas no fue descrita, sólo muestran los resultados, que a continuación se listan:

La prueba fue realizada en una PC de bajo poder, con pequeños paquetes y se muestra un resultado de la taza de transferencia de los paquetes :

- NAT64 en Linux: 9.9 kpps (*kilo packets per second*)
- NAT64 en OpenBSD: 60.2 kpps (*kilo packets per second*)

Con los paquetes grandes

- NAT64 en Linux: 84.8 Mbps (*Mega bits per second*)
- NAT64 en OpenBSD: 637.2 Mbps (*Mega bits per second*)

Los desarrolladores creen que la gran diferencia en desempeño puede ser causada por los siguientes factores:

- La implementación de Linux realiza al menos una copia adicional del paquete a la memoria, mientras que esto no ocurre en OpenBSD.
- La implementación de OpenBSD se integra con la búsqueda (*matching*) de sesiones y la tabla de estados de *pf*. Por lo tanto hereda sus mecanismos muy afinados. Por otro lado, la implementación de Linux utiliza una tabla de estado completamente separada de Netfilter, que se implementa usando un *Red-Black Tree*.
- La implementación de Linux utiliza un dispositivo virtual "NAT64" cuyo uso podría inducir una latencia adicional.

De acuerdo a (Hazeyama et al., 2011), en una comparación de varias implementaciones de NAT64 (*stateful* y *stateless*), se observó que la implementación de Ecdysis truncaba los datos de los paquetes TCP, es decir, el tamaño de los datos en el encabezado de TCP era cortado por el traductor. Por ello decidieron usar otra implementación cumpliera con todos sus requerimientos, la cuál fue linuxnat64 de Lithuania.

Dentro del estudio descrito en (Yu & Carpenter, 2012) del cual ya se habló anteriormente, los autores discuten los resultados al medir el RTT en su ambiente de pruebas y plantean la pregunta de si realmente importaría en la práctica, la diferencia entre los mecanismos estudiados. Argumentan que el RTT para una petición y respuesta HTTP tarda de 10 ms a 200 ms, dependiendo de la situación. Pero un bajo desempeño significa un incremento en la mediana del RTT, y en el caso de la traducción con NAT64 el tiempo incrementaba 3 ms. Esto no significa que la experiencia del usuario cambie, si no que refleja el tiempo de procesamiento que el dispositivo de traducción tarda y si se quiere operar un sistema como lo es NAT64 no debe convertirse en un cuello de botella.

Estudio 2: Compatibilidad de aplicaciones

Para comparar NAT64 con otros diferentes mecanismos de transición, otra opción es comparar su comportamiento en un ambiente real. Investigadores describen en (Deng,

Wang, Zheng, Gu, & Burgey, 2011) diversas pruebas hechas a algunos mecanismos de transición, entre ellos NAT64, DS-Lite, *Dual Stack*. Sus pruebas involucran el comportamiento de aplicaciones en términos de consumo de puertos/sesiones y la compatibilidad con las aplicaciones dentro de la red.

En el caso de la prueba de consumo de puertos utilizaron diferentes navegadores como, Internet Explorer y Mozilla Firefox para observar el comportamiento mientras se navegaba por diferentes páginas web. Aplicaciones como Google Earth, BitTorrent y Skype también fueron probadas.

La prueba de compatibilidad se realizó específicamente para NAT64 y se puede observar claramente en sus resultados que algunas aplicaciones simplemente no funcionan, como es el caso de aplicaciones de mensajería instantánea, aplicaciones que usan el protocolo BitTorrent y páginas web de videos. Programas como Skype, Google Earth y navegadores como los ya mencionados resultaron compatibles. Los autores proponen que una red que use *Dual Stack* en conjunto con NAT64 puede mejorar la compatibilidad de las aplicaciones, ya que algunos programas dependen de una versión de IP por lo que habrá fallas en la técnica de traducción que use NAT64.

Estudio 3: NAT64 vs. NAT44

Por otro lado, se podría comparar NAT64 contra el NAT tradicional, también conocido como NAT44. Los autores de (Llanto & Yu, 2012), documentan sus resultados al comparar el desempeño de IPv6 nativo, NAT64 y NAT44. Utilizaron ambientes controlados y “de producción” para observar el comportamiento de la red. En el ambiente de prueba se utilizaron herramientas como ping y ApacheBench, para obtener el RTT y el rendimiento (*throughput*) de un servidor web, en este caso Apache2. En el ambiente real se utilizaron herramientas como ping y algunas aplicaciones o comandos de SNMP. Por ejemplo, se usó *Multi Router Traffic Grapher* (MRTG) para graficar los resultados. Con la ayuda de estas herramientas se obtuvieron el RTT, la utilización del CPU y utilización de la memoria.

La prueba principal dentro de su ambiente consistió en usar ping/ping6 para mandar 21 paquetes al servidor para probar la conectividad, mientras que ApacheBench fue utilizado para poner a prueba al servidor con 1000, 2000 y 3000 peticiones hacia una página web de 173,225 bytes a lo largo de los niveles de concurrencia de 10, 100, 200 y 300.

Después de obtener los resultados de NAT44 y NAT64, se realizó una prueba de t-Student obteniendo para todos los resultados, excepto para la razón de transferencia, que no hay una diferencia significativa para dicha métricas. La prueba en el ambiente real consistió en usar ping y SNMP durante 15 días constantemente mientras MRTG graficaba los resultados; esta prueba se realizó dos veces, una para NAT44 y otra para NAT64. Los resultados fueron favorables para NAT64, ya que usó menos CPU, menos memoria durante la prueba y si había una diferencia significativa en la prueba de t-Student. Los resultados del RTT fueron muy parecidos entre NAT64 y NAT44.

Estudio 4: Pruebas con protocolos

Una forma práctica de probar NAT64 se describe en (Skoberne & Ciglaric, 2011), donde se usa la versión Linux de Ecdysis para realizar su estudio. Primero se seleccionó un conjunto representativo de protocolos, a nivel aplicación, que se usan en Internet. El conjunto se analizó, se evaluó y finalmente se documentaron los resultados. Éstos resultados fueron comparados posteriormente, con el resultado que en teoría deberían esperar para cada protocolo. En caso de que el protocolo no funcionara como era esperado, se analizaba una captura de paquetes para encontrar la razón.

El experimento se realizó en un ambiente de pruebas con máquinas Linux. El conjunto de protocolos trata de tomar en cuenta diferentes tipos de usuarios, como lo son los móviles, caseros y corporativos. Para evaluar dichos protocolos se seleccionaron diversas aplicaciones, la lista de protocolos que se evaluaron es la siguiente:

- BitTorrent
- FTP
- HTTP
- HTTPS
- IMAP
- NTP
- POP3
- RDP
- Skype
- MSNP
- SIP
- CIFS
- SMTP
- SSH
- TELNET
- VPN
- IPsec
- PPTP

Las aplicaciones fueron clasificadas en tres categorías: bien traducidas, condicionalmente traducidas y mal traducidas. Las aplicaciones que son traducidas condicionalmente se refieren a protocolos que necesitan de un servidor externo (Application Level Gateway) o de alguna modificación a nivel aplicación, algunas de estas aplicaciones son BitTorrent y FTP. Las otras dos clasificaciones se refieren sólo a si funcionó o no la traducción. La traducción no funcionó para Skype, MSNP, SIP, VPN, IPsec y PPTP debido, a que IPv6 no es soportado por la aplicación o que el protocolo de capas menores no es soportado por el NAT64.

Estudio 5: Análisis de paquetes / uso cotidiano

Tal vez la manera más simple de analizar el funcionamiento de NAT64 se describe en (Lähteenmäki, 2011), donde se analiza un sistema NAT64 desarrollado por LM Ericsson. Dicho sistema se probó usando dos configuraciones diferentes, una red IPv6

conectada a una red IPv4, como funciona NAT64 usualmente, y otra configuración con Dual Stack. Esto se hizo porque NAT64, en teoría, debe funcionar con las dos configuraciones. Las computadoras dentro de la red incluían sistemas operativos como Debian y Windows 7. La primera fase de pruebas se realizó con ping y traceroute. La segunda fase del proceso incluyó navegar por Internet en sitios web aleatorios y ver videos en YouTube. Y por último, la tercera fase del proceso incluyó analizar paquetes con Wireshark y el uso del comando netstat para los protocolos: ICMP, TCP, UDP, HTTP, SSH, IMAP, RTMP, FTP, además de algunos juegos y aplicaciones p2p.

En los resultados obtenidos, según el autor, se pudo apreciar que los problemas que se encontraron no estaban directamente relaciones con los protocolos o el NAT64, sino que el diseño o implementación de la aplicación. Una vez habiendo recolectado los resultados se creó una clasificación en base a los problemas encontrados. Las categorías en las cuáles se categorizaron los resultados son: direcciones literales, las direcciones dentro de los datos de los paquetes y trackers en aplicaciones p2p. También se hicieron pruebas con un punto de acceso inalámbrico, Windows 7 fue el único que funcionó sin la necesidad de configuración extra, pero Android, Mac, y un Nokia n900 si tuvieron problemas.

Estudio 6: Falla en flujos

Los autores del estudio mencionado en (Bajpai, Melnikov, Sehgal, & Schönwälder, 2012), realizaron una investigación que se enfocó en analizar las fallas de las tecnologías de transición un poco más a fondo que los estudios ya descritos anteriormente. La implementación de NAT64 probada fue Ecdysis. Como en los demás estudios, se probaron una serie de aplicaciones. Cuatro de éstas fallaron cuando tuvieron una interoperación con NAT64: Skype, OpenVPN (cliente VPN), Transmission (cliente BitTorrent) y Linphone (cliente SIP).

Según el autor, Skype no funciona con NAT64 debido a que durante el proceso de inicio de sesión el programa intenta buscar clientes en la red mandando mensajes *multicast* de DNS o mDNS, a la dirección de transporte específica *multicast* 224.0.0.251:5353. Una vez que se hace esto, inicia el proceso de autenticación con un servidor IPv4. Skype no funciona debido a que la primera acción falla en el cliente IPv6 y que la segunda acción falla por el uso del servidor DNS64. Por otro lado, OpenVPN y Transmission fallaron debido a que falta soporte de IPv6 en las aplicaciones, como ya había sido confirmado en otro estudio. La contribución más importante de este estudio es el análisis profundo para el protocolo SIP, realizado paquete por paquete usando la herramienta Wireshark. Durante el flujo de paquetes se observó que al inicio de sesión el cliente intenta conectarse a ekiga.net usando IPv6, lo que arroja un error número 606. Al recibir y al hacer una llamada el flujo parece iniciar de manera correcta, pero la llamada en sí falla debido a que los paquetes SDP contienen direcciones IPv4 literales.

Estudio 7: Análisis de sobrecarga

De acuerdo al autor de (Naito, Mori, & Kobayashi, 2012), otra manera de probar un módulo que implemente NAT64 es midiendo el rendimiento (*throughput*), la desviación estándar del rendimiento y el RTT, cuando se varía el tamaño del MSS (*Maximum Segment Size*) en conexiones TCP. Las pruebas fueron realizadas con iperf y nuttcp,

que son herramientas muy conocidas de benchmarking. El propósito de esta evaluación era confirmar la sobrecarga (*overhead*) al manipular los paquetes, debido a que la extensión o el acortamiento de un encabezado puede resultar en una sobrecarga grande. Según los autores, la sobrecarga del *throughput* depende del tamaño del MSS, porque la relación de tamaño del encabezado al tamaño total del paquete será más grande cuando el tamaño del MSS disminuya. Los resultados de las tres métricas para el módulo del *kernel* tuvieron niveles de rendimiento similares a los del *kernel* de Linux con conexiones nativas IPv4 e IPv6. Esto quiere decir que NAT64 puede lograr un alto rendimiento por tener un retraso mínimo de procesamiento. Cabe mencionar que el módulo de NAT64 probado en este estudio, es propietario, se incluyó la funcionalidad para que el protocolo SIP pudiera trabajar correctamente por medio de un ALG y algunas modificaciones en el procesamiento de los paquetes.

Análisis de Estructuras de Datos utilizadas

En esta sección se describen algunas estructuras de datos usadas por el *kernel* de Linux y que al mismo tiempo han sido usadas por las implementaciones de NAT64. Éstas estructuras de datos son de importancia porque fueron elegidas para crear las tablas de BIB y Sesión que guardan la información de la conexión dentro de NAT64. Esta sección tiene como propósito dar a conocer el análisis de cada estructura de datos para que se tengan en mente al realizar las pruebas de benchmarking.

Slab Allocator

De acuerdo a (ITESM, 2011b) hay complicaciones con el manejo y peticiones tradicionales de memoria dentro del *kernel* de Linux usando la función *kmalloc*. El manejo de memoria es similar, tanto en la implementación de linuxnat64 de Lithuania como en la etapa inicial de NAT64 de ITESM-NIC.mx y por ello los desarrolladores del proyecto NAT64 de ITESM-NIC.mx optaron por usar el caché “*slab*”, que permite un uso más eficiente y compacto de la memoria. El conflicto principal que tiene usar la función *kmalloc* surge al inicializar las estructuras para manejar las sesiones. Tradicionalmente, la memoria se pide en cantidades potencias de dos, usando un enfoque de mejor ajuste; sin embargo, esto no es lo ideal para la implementación del proyecto NAT64 de ITESM-NIC.mx dado el uso constante de estructuras cuyo tamaño no cumple con esta condición. De usarse la estructura *slab* para la asignación de memoria de la manera tradicional, se occasionaría un desperdicio de ésta a grados alarmantes, causando severas fugas de memoria. Por otro lado, la ventaja que ofrecería el uso de estructuras “*slabs*” es un uso de memoria de una manera compacta y eficiente, y sin fragmentación.

Hash list

Por otro lado, en las implementaciones de NAT64 de Lithuania y la etapa posterior del ITESM-NIC.mx se usaron las listas de *hashing* (*hash lists*) ya existentes en el *kernel*, lo que contribuye a tener código más estandarizado, reducido, legible y eficiente. Las *hash lists* son listas doblemente encadenadas, especialmente usadas para implementar las listas de desbordamiento en las tablas *hashing* (*Hash tables*).

Los elementos de la *hash list* también están integrados en las estructuras de datos

como en las listas tradicionales del *kernel*, pero existe una asimetría entre la cabeza de lista y los elementos de la lista (Mauerer, 2008). Los elementos dentro de las *hash list* se encuentran doblemente encadenados, pero la cabeza de la lista se conecta con la lista por un único puntero. El final de la lista no se puede acceder en tiempo constante, pero esto se requiere raramente. De esta manera, los datos contenidos dentro de la estructura se vuelve ligeramente más pequeños, porque sólo es necesario un puntero en lugar de dos.

Red-Black Tree

Dentro del *kernel* de Linux, los *Red-Black Trees (RB Trees)* se utilizan para manejo de memoria con el fin de organizar los elementos ordenados en un árbol. Son estructuras de datos que se utilizan con frecuencia para resolver muchos problemas, ya que ofrecen una buena combinación de velocidad y complejidad de implementación. El *kernel* de Linux implementa ésta estructura de datos como una estructura estándar y se puede utilizar dentro del *kernel* mediante la librería *rbtree.h*. La implementación de NAT64 de Ecdysis de NAT64 hace uso de esta estructura de datos.

Los árboles RB son en esencia árboles binarios que se caracterizan por las siguientes propiedades:

- Cada nodo es rojo o negro.
- Cada hoja (o nodo en el borde del árbol) es negro.
- Si un nodo es rojo, sus hijos deben ser de color negro. Por lo tanto, se deduce que no puede haber dos nodos rojos consecutivos en cualquier camino de la raíz del árbol a cualquier hoja, pero puede haber cualquier número de nodos negros.
- El número de nodos negros en un camino simple de un nodo a una hoja es el mismo para todas las hojas.

Una de las ventajas de los árboles RB es que todas las operaciones importantes de los árboles (insertar, eliminar, y la búsqueda de elementos) se puede realizar en $O(\log n)$ pasos, donde n es el número de elementos en el árbol.

Hash table

Una *hash table*, o mapa hash, es una estructura de datos que utiliza una función de hashing para asignar valores únicos o “llave” que identifican a sus valores asociados. Por ejemplo, un valor llave en una base de datos podría ser un número celular, este puede identificar de manera única al nombre de una persona. En esencia, una tabla hash implementa una arreglo asociativo. La función *hashing* se utiliza para transformar la llave en un índice único (el *hash*) que identifique a un elemento de un arreglo (conocido como *bucket*), el índice dice la posición en la cuál se debe buscar el elemento correspondiente.

Actualmente no hay una implementación estándar de una tabla de hashing en el kernel de Linux aunque ésta es una estructura de datos muy común. En el kernel se proporcionan los bloques necesarios para construir una *hash table*, por ejemplo, las *hash lists* y las funciones genéricas para el cálculo de un hash (en la biblioteca *linux/hash.h*). El uso de estos bloques puede adaptarse a un fin determinado y es responsabilidad del desarrollador. La implementación de NAT64 desarrollada por la

Universidad de Liège usa una *hash table*, pero esta estructura fue desarrollada desde cero y no utiliza ninguna facilidad del *kernel*.

Análisis de estructuras de datos

A continuación se muestra un análisis de las estructuras de datos descritas anteriormente. Para la *hash table* y el *red-black tree* se muestra el caso promedio en la tabla 2 y el peor caso para el espacio de búsqueda, la búsqueda, la inserción de datos y el borrado de datos se muestra en la tabla 3. Para la *hash list*, el análisis se muestra en la tabla 4, se describe cómo es la inserción y el borrado en las tres secciones de una lista: al inicio, en medio y al final.

	Reb-black tree	Hash table
Espacio	O(n)	O(n)
Búsqueda	O(log n)	O(1 + n/k)
Insertar	O(log n)	O(1)
Borrar	O(log n)	O(1 + n/k)

Tabla 2 Complejidad en el caso promedio

	Reb-black tree	Hash table
Espacio	O(n)	O(n)
Búsqueda	O(log n)	O(n)
Insertar	O(log n)	O(1)
Borrar	O(log n)	O(n)

Tabla 3 Complejidad en el peor caso

	Hash list
Espacio	O(n)
Búsqueda	O(n)
Insertar/borrar al inicio	O(1)
Insertar/borrar al final	O(n)
Insertar/borrar en medio	Tiempo de búsqueda + O(1)

Tabla 4 Complejidad de *hash list*

En el RFC6146, que establece cómo debe funcionar NAT64, se definen las siguientes estructuras de datos:

- UDP Binding Information Base
- UDP Session Table
- TCP Binding Information Base
- TCP Session Table

- ICMP Query Binding Information Base
- ICMP Query Session Table

Cada entrada en la BIB especifica una correspondencia entre una dirección de transporte de IPv6 y una dirección de transporte IPv4. En el caso de la consulta (*Query*) BIB ICMP, cada entrada especifica una asignación entre una tupla (dirección IPv6, ICMPv6 *Identifier*) y una tupla (dirección IPv4, ICMPv4 *Identifier*).

En tablas de la sesión TCP y UDP, cada entrada especifica una asignación entre una tupla de direcciones de transporte IPv6 y una tupla de direcciones de transporte IPv4. En la tabla de consulta de sesión ICMP, cada entrada especifica una asignación entre una dirección IPv6 de origen, una dirección de destino IPv6, un identificador ICMPv6, una dirección IPv4 de origen, una dirección IPv4 de destino, y un identificador ICMPv4.

Dado que NAT64 es un ambiente de estado, necesita de estructuras de datos para manejar la información de cada conexión. La elección éstas estructuras de datos influye en el rendimiento del módulo. Hay tres aspectos que se tienen que considerar al seleccionar una estructura de datos para un módulo de red que se use en el *kernel* de Linux, que son:

- Manejo de memoria.
- Tiempo de ejecución.
- Sincronización, para evitar cierres de exclusión mutua.

Dada la importancia de estos tres aspectos, se puede inferir que los resultados de cualquier prueba de desempeño están ligados al tipo de estructura de datos usada.

La investigación del marco teórico permitió observar que NAT64 no funciona con algunas aplicaciones. Esto determinó que para realizar las pruebas de desempeño se debieran elegir una serie de aplicaciones que este probado que trabajan correctamente con NAT64.

Por otro lado, en algunos estudios se comparan algunas métricas de NAT64, NAT44, IPv4, IPv6, incluso con otros mecanismos de transición y se analizan los resultados. Es de esperarse que el uso del NAT haga más lenta la comunicación entre dos redes, debido al procesamiento extra que implica la traducción de direcciones. Las métricas comúnmente propuestas en los estudios mencionados en el párrafo anterior son: *throughput*, *Round Trip Time* (RTT), utilización de CPU, y utilización de memoria. También, se proponen otras métricas, como la desviación estándar, pero éstas no siempre son usadas para medir el mismo parámetro. Por ejemplo, en el estudio de mecanismos de transición (Yu & Carpenter, 2012) el RTT mide el tiempo de respuesta de peticiones http. Ninguno de los estudios tienen alguna métrica para medir el desempeño de la traducción de UDP. Por lo que se necesita una métrica para cada protocolo. Lo importante que se puede obtener de estos estudios son: las métricas y técnicas de pruebas. De acuerdo a este marco teórico se pretende abstraer un modelo estándar para realizar un benchmarking de desempeño, el cual se presenta en el capítulo siguiente.

Capítulo 3 Definición de la propuesta de Benchmarking para aplicaciones NAT64

Los hallazgos después de la investigación bibliográfica realizada indican que existen diversas y variadas formas de evaluar el desempeño de un sistema NAT64. Por lo que se puede decir que por el momento no existe algún **método estándar** para realizar un benchmarking de desempeño a NAT64. Por otro lado, los resultados de la investigación también permitieron concluir que actualmente no existe una comparación entre los diferentes tipos de desarrollos de NAT64, misma que pueda dar información real para evaluar las diferencias entre ellos.

Es por esto que el presente trabajo de tesis, propone un modelo de pruebas para realizar un benchmarking de desempeño a las implementaciones del NAT64.

El ambiente de pruebas utilizado en este trabajo es controlado y limita el uso a una serie de aplicaciones. Para probar NAT64 se cubren los tres protocolos de capa de red que el RFC6146 especifica. Por lo tanto, las aplicaciones elegidas para probar el desempeño tienen que usar alguno de estos tres protocolos: UDP, TCP e ICMP. Al igual que en el estudio de mecanismos de transición (Yu & Carpenter, 2012), se usará la división de paquetes en: pequeños y grandes. La razón detrás de esto surge de la experiencia obtenida a través de la participación en la implementación de ITESM-NIC.mx NAT64, y de la técnica de valores límite, que es clásica en pruebas de software.

El proceso de pruebas se dividió en dos etapas: una etapa de funcionalidad y otra etapa de desempeño. En la etapa de funcionalidad se probó la conexión, entre dos redes IPv4 e IPv6, mediante NAT64. Para ello se utilizaron herramientas que hacen uso de los protocolos de red mencionados en el párrafo anterior y que son relativamente fácil de configurar, o no necesitan configuración. Sin embargo, aunque para la etapa de desempeño también se tomaron las mismas consideraciones, las aplicaciones elegidas toman en cuenta la necesidad de poder generar y soportar un gran número de peticiones de manera simultánea. Aunque se recomienda utilizar estas aplicaciones, no se descarta la inclusión de otras diferentes.

La selección final de aplicaciones utilizadas en las pruebas se muestra en la tabla 5. Se observa que en la parte de "Protocolo capa aplicación", las aplicaciones ping6 y traceroute6 tiene un N/A, ya que este tipo de aplicaciones son sólo herramientas que usan los protocolos de red de capa cuatro.

Aplicación	Protocolo capa aplicación	Protocolo capa red
Apache2	HTTP	TCP
BIND	DNS	UDP
ntpd	NTP	UDP
ping6	N/A	ICMP
openssh	SSH	TCP
traceroute	N/A	ICMP

Tabla 5 Aplicaciones a probar

Etapa de funcionalidad

El primer paso para realizar las pruebas de funcionalidad fue configurar un ambiente de pruebas. El ambiente consta de dos redes, una red IPv4 y otra IPv6, y al menos tres máquinas, una que actúa de servidor NAT64 conectando éstas dos redes, otra que actuó como cliente IPv6 y la tercera que cumplió la función de un cliente IPv4.

El diseño de la topología simula un ambiente real, en donde los paquetes tendrán que pasar por diferentes equipos de red, como son los enrutadores y los conmutadores. Por esto, se agregaron dos enrutadores y dos conmutadores en la red. A continuación se muestra una descripción del ambiente de pruebas.

Como se puede apreciar en la Figura 1, el servidor NAT64 que conecta las dos redes tiene dos interfaces de red físicas. En el caso de los enrutadores también tienen dos interfaces de red, una de estas es conectada directamente al servidor NAT64 y la otra va conectada a un conmutador. Cabe mencionar que en el enrutador de la red IPv6 se configuró una ruta estática para que pudiera rutear los paquetes dirigidos al servidor NAT64 (con el prefijo de red 64:ff9b/96), y en el enrutador de la red IPv4 otra ruta estática que maneje los paquetes dirigidos a la pool de IPv4. En la sección I de Anexos se muestra la configuración de la infraestructura de red.

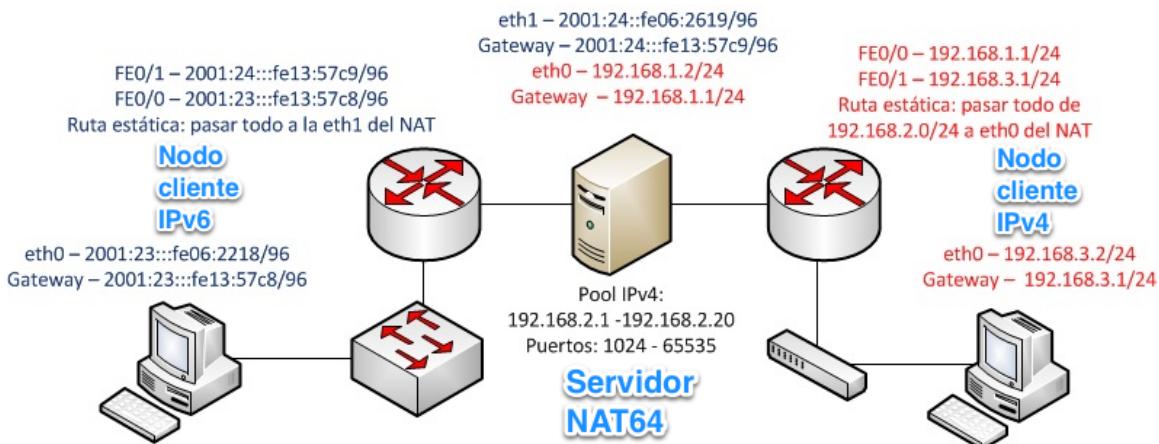


Figura 1 Ambiente de pruebas

El sistema operativo que se instaló fue Ubuntu 10.04 para el servidor NAT64 de Linux y OpenBSD 5.1 para el servidor NAT64 de OpenBSD. En el caso de los clientes IPv6 e IPv4 se usó la versión de escritorio de Ubuntu 10.04. Si bien esto no significa que los clientes no podrían usar OS X o Windows, se eligió Ubuntu por la facilidad de configuración y uso de las diferentes herramientas requeridas para las pruebas. A continuación se puede ver en la tabla 6 y 7 el detalle las versiones específicas de los sistemas operativos que se usaron. Algunas características importantes sobre el equipo de computo (*hardware*) para los clientes se pueden apreciar en la tabla 6. La tabla 7 muestra información de los nodos servidores.

Nodo cliente IPv6	Nodo cliente IPv4
<ul style="list-style-type: none"> • CPU: Intel Core 2 vPro • RAM: 2.3 GB • HDD: 1 x 140 GB • NIC: 1 (eth0) • Plataforma: 64 bits • SO: Ubuntu 10.04 Desktop 	<ul style="list-style-type: none"> • CPU: Intel Pentium 4 • RAM: 1 GB • HDD: 1 x 120 GB • NIC: 1 (eth0) • Plataforma: 64 bits • SO: Ubuntu 10.04 Desktop

Tabla 6 Hardware para clientes

Nodo servidor Linux	Nodo servidor OpenBSD
<ul style="list-style-type: none"> • CPU: Intel Core 2 vPro • RAM: 2.3 GB • HDD: 1 x 140 GB • NIC: 2 (eth0 & eth1) x 100Mbps • Plataforma: 64 bits • SO: Ubuntu 10.04 Server 	<ul style="list-style-type: none"> • CPU: Intel Core 2 vPro • RAM: 2.3 GB • HDD: 1 x 140 GB • NIC: 2 (eth0 & eth1) x 100Mbps • Plataforma: 64 bits • SO: OpenBSD 5.1

Tabla 7 Hardware para servidores

Entre el equipo de red utilizado debe contarse con enrutadores que soporten IPv6 e IPv4. El equipo de red que fue usado para crear la topología es el siguiente:

- 2 x *Router Cisco 2800 Series Integrated services. IOS version 12.4*
- 1 x *Switch Cisco Catalyst 2960 series*
- 1 x *8-port Workgroup Switch Cisco Linksys*

La configuración de los enrutadores Cisco se incluye en la sección III de Anexos. Los comutadores no requirieron de alguna configuración.

Una vez que se instaló el software indicado y se configuró la red, se comenzó con las pruebas de la etapa de funcionalidad. Durante esta etapa de prueba se utilizaron tres aplicaciones: SSH, NTP y traceroute6 (en modo ICMP), con las que se probó la funcionalidad del nat64 cuando se utilizan los protocolos TCP, UDP e ICMP, respectivamente. Después de probar que la correcta comunicación en cada protocolo se procedió a realizar las pruebas de desempeño.

Etapa de desempeño

Para realizar el benchmarking de desempeño, se definieron una serie de métricas para apreciar los resultados de las pruebas. Dichas pruebas involucraron a aplicaciones para

probar los tres protocolos, estas fueron Apache, BIND y ping6. Se utilizó un tamaño de paquete diferente para cada tipo de prueba, asegurando que el campo de datos (*payload*), dentro de los paquetes, estuviera dentro del rango deseado. El tamaño de los paquetes de prueba se categorizó en 2 diferentes tipos:

- Paquetes pequeños. Paquetes con un tamaño de datos de 120 bytes.
- Paquetes grandes. Paquetes con un tamaño de datos de 1200 bytes

Las métricas definidas para evaluar el desempeño de cada implementación de NAT64 que se probaron son las siguientes:

- Rendimiento (*throughput*). El rendimiento se define como el número de procesos que son completamente terminados en un cierto intervalo o unidad de tiempo. Por ejemplo, el rendimiento de HTTP puede proporcionar una estimación realista en cuanto a la experiencia al navegar por Internet. Representa la rapidez en la que una página web puede ser cargada por el navegador web. Es una métrica de gran importancia porque la navegación web es uno de los servicios de banda ancha más utilizados. La medición se llevan a cabo mediante la solicitud de una página web y observando el tiempo requerido para descargar la página web. El rendimiento se puede calcular de la siguiente manera:

$$Th = \frac{TWS}{S} \text{ KB/s}$$

Ecuación 1 Fórmula para calcular el rendimiento de HTTP

Donde TWS el número total de bytes descargados de la página web y S es el tiempo en cual se descargaron las peticiones.

- *Round Trip Time* (RTT) promedio. El tiempo transcurrido entre el envío de la petición y la recepción de la respuesta, es el tiempo de ida y vuelta. Es una medida directa de la latencia entre los dos extremos. El RTT se puede calcular fácilmente con ICMP al medir el tiempo que tarda en llegar un mensaje ECHO REPLY después de haber enviado un mensaje ECHO REQUEST. El RTT promedio se puede calcular de la siguiente forma:

$$L = \frac{1}{N} \sum_{i=1}^N L_i \text{ ms.}$$

Ecuación 2 Fórmula para calcular el RTT promedio de ICMP

Donde N es el número de peticiones y Li es la latencia de cada petición i.

- Latencia promedio de DNS. Aunque las peticiones DNS, en un ambiente con NAT64, deben ser procesadas por el servidor DNS64, en este caso se utilizó este tipo de peticiones una fuente de generación de paquetes. El tiempo de respuesta o latencia de DNS se mide mediante la petición de una consulta DNS. El cálculo de latencia se mide con la diferencia tiempo transcurrido entre el envío de la consulta y la recepción de la respuesta. Este ciclo se repite N veces y el tiempo de DNS de respuesta se calcula como la media de las N mediciones, como se puede ver en la siguiente fórmula.

$$DRT = \frac{1}{N} \sum_{i=1}^N Ti \text{ ms.}$$

Ecuación 3 Fórmula para calcular la latencia promedio en DNS

Donde N es el número de peticiones y Ti es la latencia de cada petición i.

- Utilización de CPU. Es el tiempo que un proceso está en el estado de corriendo. (Utilizando el CPU). Para calcular la utilización de CPU se utilizó la información del sistema.
- Utilización de memoria. Es el porcentaje de memoria utilizada por una aplicación o por un sistema de cómputo. Para calcular la utilización de memoria se utilizó la información del sistema.

Para capturar o evaluar estas métricas se utilizó el software de la tabla 8. Se describen a continuación:

- **Observium** es un sistema de monitoreo y observación de red desarrollado en PHP/MySQL, que colecta datos de los dispositivos que usan SNMP y lo presenta a través de una interfaz web.
- **ApacheBench** es una herramienta utilizada para la evaluación comparativa de *Hyper-Text Transfer Protocol* (HTTP). Está diseñado para proporcionar información sobre el rendimiento del servidor Apache. Se permite la simulación de distintas cantidades de peticiones y niveles de concurrencia.
- **dnsperf** es una herramienta de prueba de rendimiento para el servicio DNS.

Métrica	Aplicación
Throughput HTTP	Apache Bench
Latencia DNS	DNSperf
RTT de ICMP	Ping
Utilización de CPU	Observium
Utilización de memoria	Observium

Tabla 8 Herramientas para pruebas

Para cada tipo de paquete se realizó una serie de pruebas variando la cantidad de peticiones a procesar por el NAT64. A continuación se define la serie de pruebas desarrolladas con cada aplicación.

- Para Apache se harán 100 peticiones con un nivel 10 de concurrencia, después 200 peticiones con un nivel 20 de concurrencia, y finalmente 400 peticiones con un nivel 40 de concurrencia. Las peticiones se hacen una tras otra. El nivel de concurrencia determina el número de hilos que se usan para simular usuarios concurrentes.
- Para BIND se realizarán 100 peticiones, después 200 peticiones y finalmente 400 peticiones. Las peticiones se hacen una tras otra.
- Para ping se harán 10 peticiones, después 20 peticiones y para finalizar 40 peticiones. Las peticiones se hacen una tras otra.

El modelo propuesto de benchmarking para aplicaciones NAT64 se puede resumir en los pasos descritos en la figura 2. Al iniciar se prueban 3 aplicaciones de la etapa de funcionalidad, si alguna de ellas falla, se termina el benchmarking para ese NAT64. De lo contrario se inicia etapa de desempeño en donde se obtienen las métricas. Durante todo el proceso se obtienen métricas de uso de memoria y de CPU.

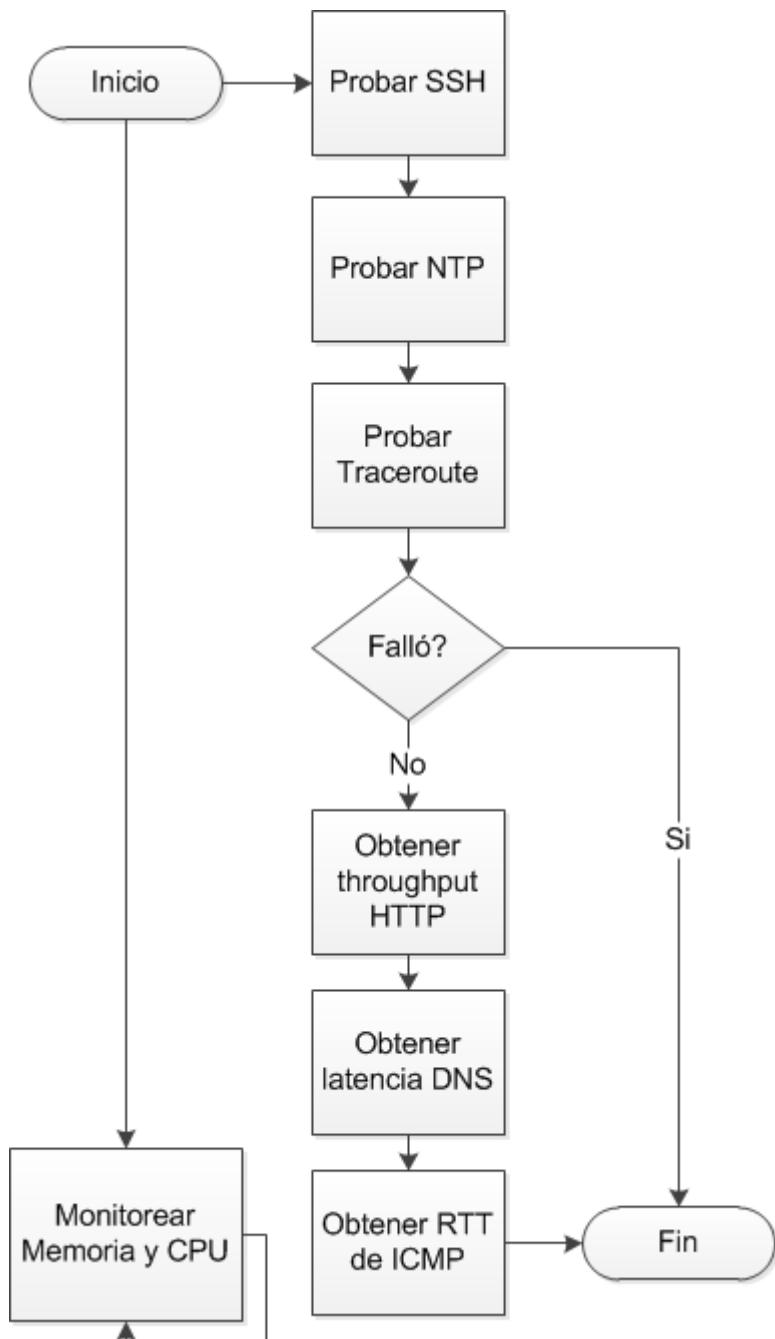


Figura 2 Modelo de benchmarking para aplicaciones NAT64

Experimento

Se desarrolló un pequeño prototipo de prueba de concepto. La prueba de concepto tiene como propósito hacer un cambio a la aplicación NAT64 de ITESM-NIC y utilizar el benchmarking propuesto para medir el rendimiento y saber si el benchmarking es capaz de usarse como una manera de detectar problemas de diseño de software. Al interpretar los resultados se quiere saber por qué el cambio hizo mejorar o empeorar al sistema.

Prueba de concepto

La prueba de concepto consiste en cambiar la manera en la cual el módulo recibe y envía paquetes. Sistemas como Ecdysis, linuxnat64 de Lithuania y TAYGA usan una interfaz virtual para mandar y recibir paquetes en la red. La manera actual en que ITESM-NIC recibe paquetes es un HOOK de Netfilter, es decir, mediante una regla de iptables. Para mandar paquetes se utiliza la función `ip_local_out()` para IPv4 y la función `ip6_local_out()` para IPv6. En si se añadió código para usar un `netdevice` en el módulo del `kernel` y una vez instalado se levanta un interfaz virtual y se añaden rutas para IPv4 e IPv6 mediante comandos en la terminal.

Los cambios al código se encuentran en ITESM-NIC-2, la sección II de los Anexos muestra como tener acceso a este nuevo código. Durante el desarrollo del prototipo también se eliminó la dependencia del módulo de **conntrack** por lo que en teoría reduciría el procesamiento que hace Netfilter cada vez que recibe un paquete. La desventaja de hacer esto, es que sin conntrack se tendría que re implementar el manejo de fragmentos para IPv4 e IPv6 y el proceso de determinar tupla tendría que re incluir código que el `kernel` implementa para manejo de paquetes ICMP, por ejemplo.

La premisa que se tiene al usar una interfaz virtual es que el *throughput* disminuye y la latencia aumenta en un sistema de red. Por otro lado, el uso de **conntrack** aumenta radicalmente el tiempo de procesamiento por lo que se podría decir que poner uno y quitar otro, los cambios son casi transparentes.

En este capítulo fue definida la propuesta de benchmarking para aplicaciones NAT64. Se describió a detalle el ambiente de pruebas controlado a utilizar. El benchmarking consiste en dos etapas: funcionalidad y desempeño. Para éste se eligieron una serie de aplicaciones a probar junto con una serie de métricas a medir. Se estableció que se usarían dos tipos de paquetes: pequeños y grandes. Finalmente se describió la serie de pruebas a realizar durante éstas dos etapas y se describió una prueba de concepto. En el siguiente capítulo se muestran los resultados.

Capítulo 4 Resultados

Este capítulo presenta los resultados para cada etapa de la propuesta de benchmarking definida en el capítulo anterior. Después de terminar la definición de las pruebas, se instalaron y configuraron las siguientes aplicaciones de NAT64 en el nodo servidor de Linux y OpenBSD respectivamente:

- Ecdysis.
- linuxnat64 de Lithuania.
- TAYGA.
- OpenBSD NAT64.
- ITESM-NICmx

Resultados de la etapa de funcionalidad

En esta etapa se realizaron las pruebas de funcionalidad para verificar la conexión entre la red IPv6 y la red IPv4 por cada aplicación. En la tabla 9 se resumen los resultados, donde se observa que todas las aplicaciones NAT64 pasaron de manera exitosa esta prueba.

Aplicación	SSH	NTP	TRACEROUTE
Ecdysis	✓	✓	✓
Lithuania	✓	✓	✓
TAYGA	✓	✓	✓
OpenBSD	✓	✓	✓
ITESM-NIC	✓	✓	✓

Tabla 9 Resultados de funcionalidad

La prueba de SSH consistió en usar el siguiente comando:

```
ssh cliente2@64::ff9b::192.168.3.2
```

Para la prueba de traceroute6 se usó el siguiente comando:

```
traceroute -6 -I 64::ff9b::192.168.3.2
```

Finalmente para la prueba de NTP se corrió el siguiente comando:

```
ntpdate 64::ff9b::192.168.3.2
```

Al ver que todos pasaron la prueba de funcionalidad se inició la etapa de pruebas de desempeño.

Resultados de la etapa de desempeño

En esta sección se muestran los resultados de la etapa de desempeño descrita en el capítulo anterior. En general se hicieron dos pruebas para cada protocolo: UDP, TCP e ICMP. Estas pruebas se realizaron una tras otra para cada sistema y se guardaron los resultados para después graficarlos y compararlos como se describe a continuación.

Pruebas del protocolo TCP

La primera prueba consistió en verificar el *throughput* del NAT64 cuando se utiliza el protocolo TCP, para esto se hicieron múltiples peticiones a un servidor Apache. Las peticiones de paquetes pequeños se hicieron a la página index.html, que viene incluida con la instalación por defecto de Apache, con un tamaño de 127 bytes. Para la prueba de paquetes grandes se creó una página HTML con el contenido suficiente para incrementar su tamaño a 1216 bytes.

Como ya se había mencionado, para realizar las pruebas de TCP, se instaló el servidor web Apache2 en la máquina IPv4 y se usó ApacheBench para mandar peticiones HTTP desde la máquina cliente IPv6. ApacheBench necesita como parámetros el número de peticiones, el nivel de concurrencia y la URL a visitar. Un comando ejemplo para usar esta herramienta se muestra a continuación:

```
ab -n 100 -c 10 http://[64:ff9b::192.168.3.2]/
```

Donde *-n* es el número de peticiones *-c* es el nivel de concurrencia y después viene la URL. Usando esta herramienta se hicieron tres pruebas de manera secuencial para los diferentes tipos de paquetes. La descripción de la prueba se muestra en la tabla 10.

	Número de peticiones	Usuarios concurrentes
Corrida 1	100	10
Corrida 2	200	20
Corrida 3	400	40

Tabla 10 Prueba de desempeño para TCP

Para definir un grado de confianza en los resultados se utilizó la prueba de **t-student**. Cada prueba de los experimentos se repitió 10 veces para obtener al menos un grado de confianza de 90%. De acuerdo a (Hassan & Jain, 2004), la meta de la simulación es obtener la media de una variable particular, y como no es posible obtener un valor exacto, las simulaciones estocásticas tratan de estimar la media de un número de observaciones para el caso de un número de diferentes secuencias. El autor también menciona que la precisión relativa de una simulación usualmente utiliza un nivel de confianza de 95% y se debería tratar de converger a este nivel realizando el número de pruebas necesarias. Pero en este benchmarking, un nivel de confianza de 90% fue elegido ya que se necesitaba comparar todos los sistemas en un mismo nivel de confianza. Aunque muchos de los resultados si permitían un nivel de confianza de 95%, en otros sistemas no se lograba, por lo que se redujo en un 5% el nivel para evitar hacer más pruebas y lograr que todos los sistemas convergieran en N pruebas.

El valor esperado o precisión relativa para la prueba de t-student se calculó usando la siguiente fórmula:

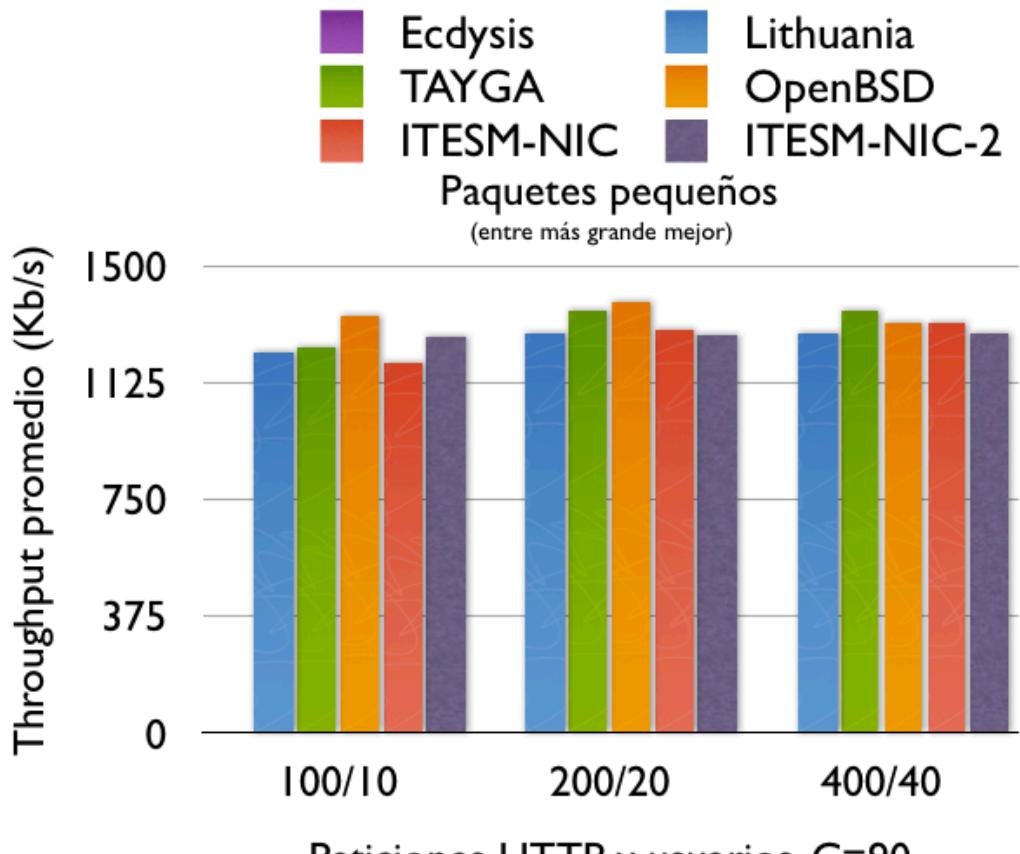
$$\varepsilon = \frac{t_{n-1,1-\frac{\alpha}{2}} * \sigma[\bar{x}(n)]}{\bar{x}(n)}$$

Ecuación 4 Fórmula para calcular el nivel de confianza

Donde, n es el número de pruebas realizadas, en este caso es 10. Por lo que $t_{n-1,1-\frac{\alpha}{2}}$ es igual a 1.8331. La variable sigma es la raíz cuadrada de la varianza y \bar{x} es la media. De acuerdo a (Hassan & Jain, 2004) si ε es menor que 0.1 se tiene un resultado con una confianza de 90%.

A continuación se muestran los resultados de la prueba de t-student, donde todos tienen al menos un grado de confianza de 90%. Los resultados se resumen en gráficas de barras, que muestran el promedio de los 10 resultados de cada prueba. Durante la prueba de TCP, Ecdysis continuaba presentando errores de *kernel panic*, por lo que fue omitido en esta ronda de pruebas.

La Figura 3 muestra los resultados de la prueba con paquetes pequeños. En ella se observa un buen desempeño de la versión de NAT64 de Lithuania y de OpenBSD. En el caso de OpenBSD, se piensa que los resultados tienen estas características debido a la gran integración con el sistema pf, la rapidez de pf se puede apreciar claramente por el hecho de haber procesado el mismo número de peticiones en un tiempo muy corto. Por otro lado, si se toma en cuenta que TAYGA es una aplicación en userspace, es muy rápida, la combinación de NAT44 y TAYGA los hace una buena opción en el caso de desempeño de TCP.



Peticiones HTTP y usuarios, C=90
Figura 3 Resultados TCP de paquetes pequeños

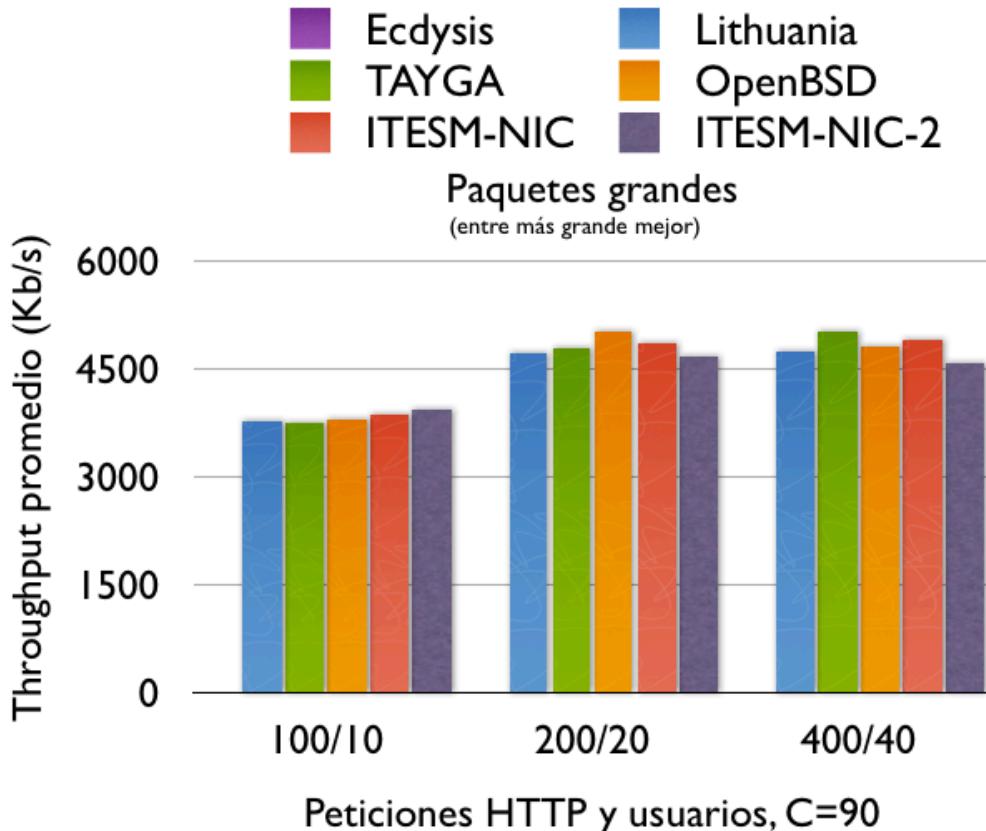


Figura 4 Resultados TCP de paquetes grandes

La Figura 4 muestra los resultados de la prueba con paquetes grandes. En ella se observa un buen desempeño de la versión de NAT64 de TAYGA, OpenBSD e ITESM-NIC. Se podría decir que los códigos optimizados de OpenBSD y NAT44 usado por TAYGA, son un factor que les hace tener un mejor desempeño tanto en cargas pequeñas de datos como en cargas grandes.

Pruebas del protocolo UDP

En esta sección se muestran los resultados de la serie de pruebas para el caso de paquetes pequeños. En donde se midió la latencia, mediante el uso de peticiones DNS. Para esto se instaló el servidor BIND en el nodo cliente IPv4 y se configuró de tal manera que pudiera resolver la URL example.com. La herramienta que se utilizó para mandar peticiones DNS a través del servidor NAT64 fue DNSperf. Esta toma como parámetro la familia IP, un archivo con las consultas a realizar y la dirección del servidor DNS. Un ejemplo de como usar esta herramienta se muestra a continuación:

```
dnsperf -f inet6 -d dns-test -s 64:ff9b::192.168.3.2
```

Donde el parámetro *-f* indica la familia IPv6, la opción *-d* indica el nombre de un archivo que contiene las URLs de los nombres de dominio que se consultan, y finalmente la opción *-s* es la dirección IP del servidor DNS.

Por otro lado, para la prueba de paquetes grandes se configuró DNSSEC en el servidor BIND, máquina tal, para que resolviera el domino example.org. De esta manera la respuesta de DNS pudo ser de tamaño grande, mayor a 1100 bytes. Se añadieron dos opciones más al comando de DNSPerf. Por ejemplo, se puede usar esta herramienta de la siguiente forma:

```
dnsperf -f inet6 -d dns-test -s 64:ff9b::192.168.3.2 -e -D
```

Donde la opción *-e* indica que se usarán extensiones de DNS y la *-D* indica el uso de DNSSEC. Usando esta herramienta se hicieron tres pruebas de manera secuencial para los diferentes tipos de paquetes. La descripción de esta prueba se muestra en la tabla 11.

Número de peticiones

Corrida 1	100
Corrida 2	200
Corrida 3	400

Tabla 11 Prueba de desempeño para UDP

La Figura 5 muestra los resultados de la prueba con paquetes pequeños. En ella se observa un buen desempeño de la versión de NAT64 de ITESM-NIC, ITESM-NIC-2 y de OpenBSD. Esta es una prueba clave que cumple uno de los objetivos específicos de esta tesis, el cual es que el benchmarking sea capaz de mostrar diferencias en desempeño entre dos mismas implementaciones de NAT64 pero con ligero cambios en el diseño. Este tipo de pruebas podrá ayudar a los desarrolladores a elegir una estrategia de diseño por ejemplo, de cómo recibir/mandar paquetes que influya directamente al desempeño de la aplicación. Los resultados obtenidos se muestran en las siguientes gráficas.

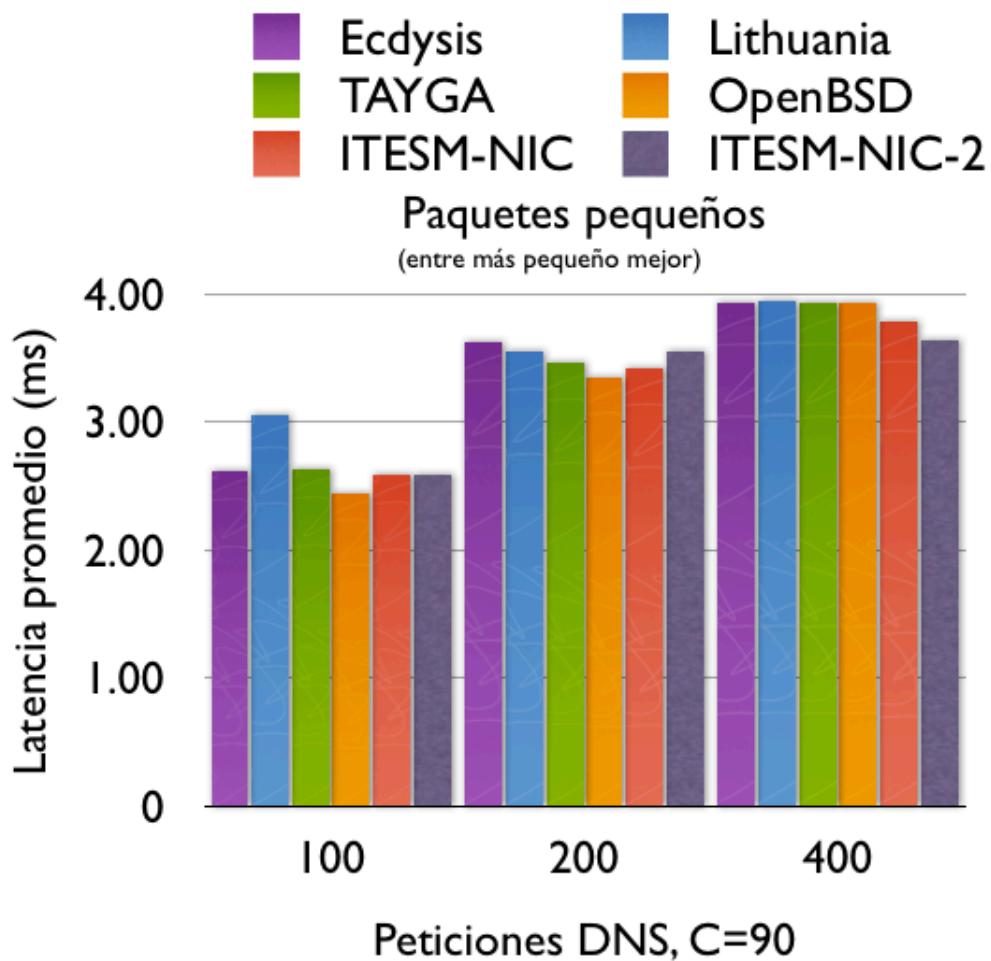


Figura 5 Resultados UDP paquetes pequeños

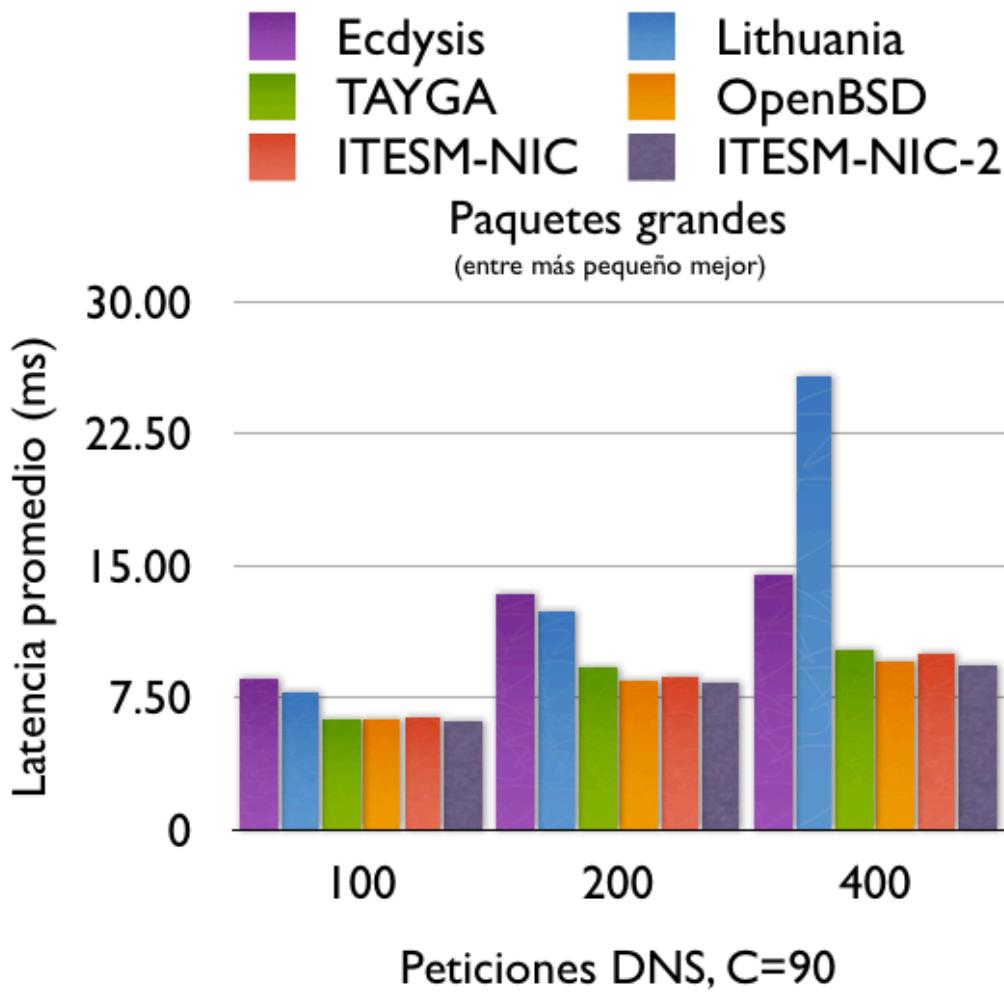


Figura 6 Resultados UDP paquetes grandes

La Figura 6 muestra los resultados de la prueba con paquetes grandes. En ella se observa un buen desempeño de la versión de NAT64 de ITESM-NIC, ITESM-NIC-2 y de OpenBSD. Como los tiempos de retraso en el red son mínimos por tener un ambiente controlado, se podría decir que el retraso que se puede apreciar de parte de Ecdysis y Lithuania se podrían deber a la manera en la cuál hacen la traducción del paquete ó por su estructura de datos.

Pruebas del protocolo ICMP

Finalmente, se realizaron las pruebas para medir el RTT al utilizar el protocolo ICMP. Para ello se utilizó la herramienta ping. Usando esta herramienta se hicieron tres pruebas de manera secuencial, para los diferentes tipos de paquetes. La descripción de la prueba se encuentra en la tabla 12.

Número de peticiones	
Corrida 1	10
Corrida 2	20
Corrida 3	40

Tabla 12 Prueba de desempeño para ICMP

En el caso de las pruebas con paquetes pequeños, el único parámetro extra que se indicó a la herramienta fue el número de peticiones. Un ejemplo de la línea de comandos utilizada es la siguiente:

```
ping6 -c 10 64:ff9b::192.168.3.2
```

Donde `-c` indica el número de peticiones ECHO REQUEST a mandar, y por último se indica la dirección de la máquina que recibirá estas peticiones.

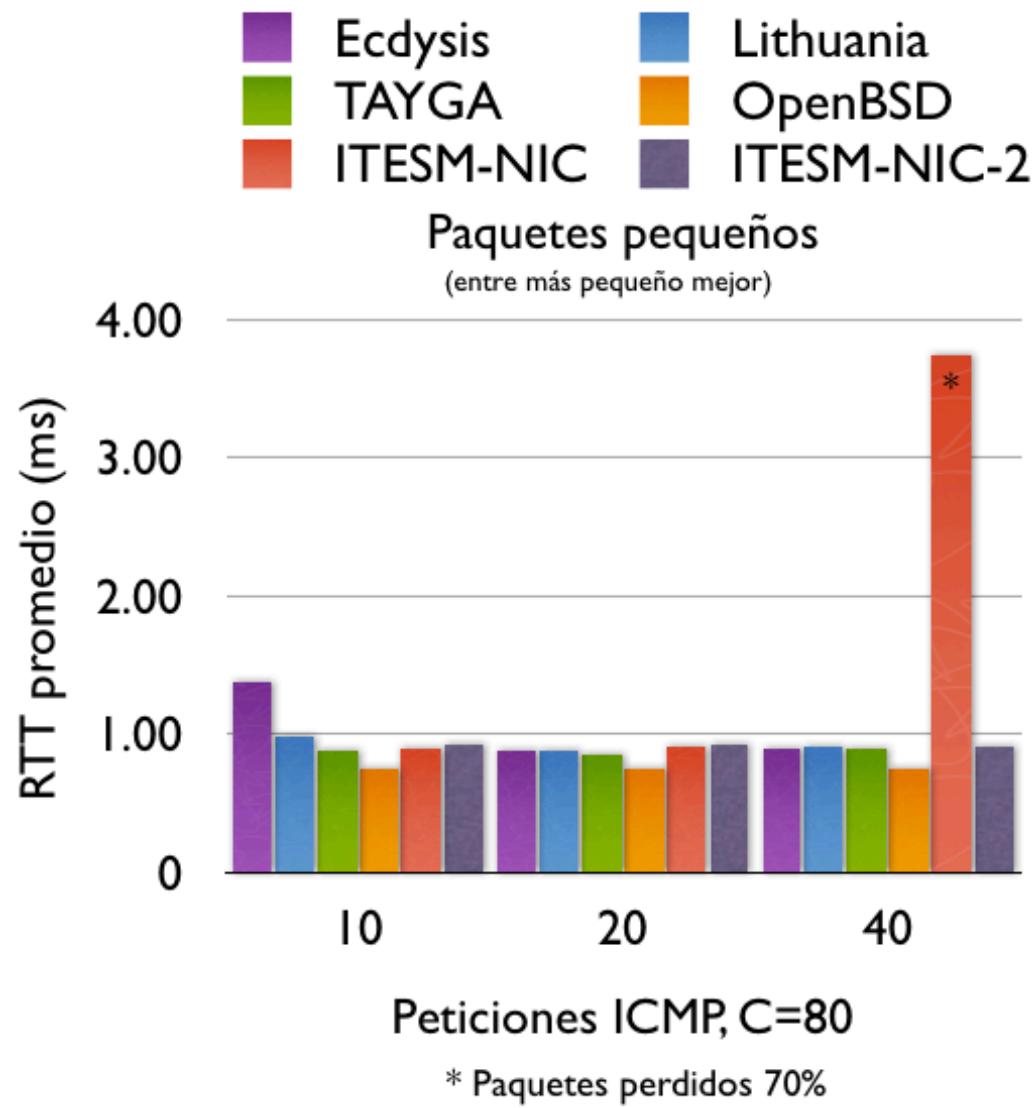


Figura 7 Resultados ICMP paquetes pequeños

La Figura 7 muestra los resultados de la prueba con paquetes pequeños. En ella se observa un buen desempeño de la versión de NAT64 de Lithuania y de OpenBSD. Aquí se tiene un porcentaje de paquetes perdidos del 70% de parte de ITESM-NIC. ITESM-NIC-2 no tiene esos problemas al usar una interfaz virtual para recibir/mandar paquetes pero aún hay lugar para mejoras, sin embargo si logra ser más competitivo. Es

importante mencionar que la implementación de la traducción del protocolo ICMP en el sistema NAT64 de ITESM-NIC no está completo y faltan algunas funcionalidades de acuerdo al RFC6146, por lo que razón exacta del problema se desconoce. Por otro lado, el benchmarking puede detectar el problema mencionado por (Kriukas, 2010), quien detalla los problemas que puede generar el no usar una interfaz virtual para el intercambio de paquetes.

En el caso de las pruebas con paquetes grandes se añadió el parámetro “-s 1200”, a la lista de parámetros de la herramienta, para mandar un mensaje con tamaño de datos de 1200 bytes. Se muestra a continuación un ejemplo:

```
ping6 -c 10 64:ff9b::192.168.3.2 -s 1200
```

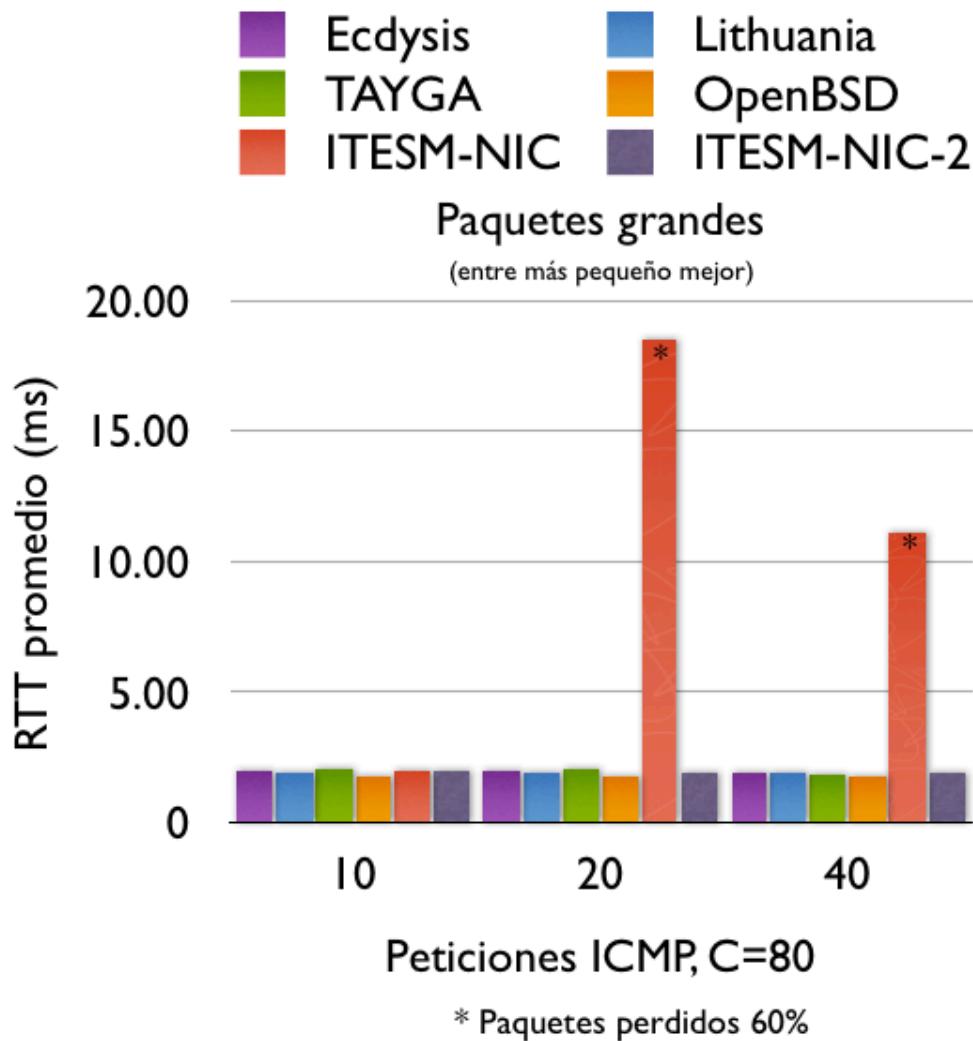


Figura 8 Resultados ICMP paquetes grandes

La Figura 8 muestra los resultados de la prueba con paquetes grandes. En ella se observa un buen desempeño de la versión de NAT64 de Lithuania y de OpenBSD. En el caso de OpenBSD, se piensa que los resultados que tiene le han favorecido hasta ahora a este sistema en la mayoría de los ámbitos. En este punto se podría decir que

dentro de todos sistemas de NAT64, OpenBSD tiene el mejor desempeño en promedio. ITESM-NIC sigue teniendo problemas con tiempos de latencia muy altos.

Observaciones

Durante las pruebas de ICMP, se notó un problema con el sistema NAT64 del ITESM-NIC. Después de analizar con más cuidado al sistema, se puede decir que tenía dos problemas:

1. Paquetes perdidos.
2. Tiempo de respuesta muy alto.

Se pudo observar que estos sucesos son mutualmente excluyentes, es decir cada uno de estos problemas sucedieron de manera independiente. Sin embargo, el problema que causa este comportamiento podría ser el mismo.

Durante las pruebas, se estuvo monitoreando constantemente al servidor NAT64 con la herramienta Observium, vía el protocolo SNMP. Previo a las pruebas, se configuró el cliente IPv6 para que la herramienta Observium mandara peticiones SNMP y obtuviera información del sistema. Los servidores NAT64 también fueron configurados para que trabajaran con Observium, se instaló la herramienta Net-SNMP que sirve como cliente SNMP. Observium se encargó de guardar información sobre el uso de CPU y memoria, una serie de gráficas se presentan a continuación.

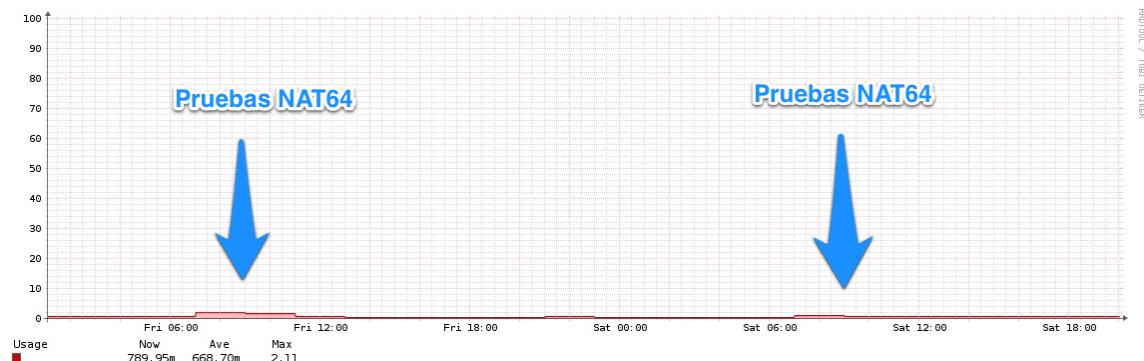


Figura 9 Uso de CPU



Figura 10 Uso de memoria física

La figura 8 muestra el uso de CPU cuando se estaba utilizando la traducción de NAT64 y cuando el sistema estaba libre de uso. La figura 9 muestra el uso de memoria cuando se estaba utilizando la traducción de NAT64 y cuando el sistema estaba libre de uso. No se pudieron observar picos en el CPU o en la memoria. Por otro lado, se puede decir que en general las aplicaciones de NAT64 tienen un buen manejo de memoria, y el procesamiento de los paquetes no influye tanto en el uso de CPU. Básicamente el uso de CPU, memoria virtual o física es casi despreciable.

Análisis de la prueba de concepto

Con los resultados de ITESM-NIC-2 se puede probar que la interfaz virtual aumenta la latencia para ICMP, UDP y TCP y por esto es que el *throughput* disminuye, pero el hecho de que se elimina el uso de **conntrack** no afecta tanto al sistema. Sin embargo el uso de **conntrack**, desde el punto de vista de un servidor es indispensable para un sistema NAT. El procesamiento que se le hace al paquete a través del *stack* de TCP/IP puede ser pesado que el procesamiento que hace **conntrack**. Sin duda hay mejoras en UDP y en ICMP que obligan a tomar en consideración el uso de una interfaz virtual para enviar/recibir paquetes en el servidor NAT64.

Aunque claramente hay un *trade-off* en la manera de recibir y mandar paquetes en la prueba de concepto realizada, la propuesta de benchmarking es capaz de mostrar resultados valiosos y que se pueden analizar con facilidad. Por esto, se puede resumir que la prueba de concepto fue un éxito en probar que se puede usar el benchmarking propuesto en este documento para medir cambios hechos a un sistema NAT64 y compararlo contra otros ya existentes.

Análisis de resultados

A continuación se presenta el análisis de los resultados del benchmarking para aplicaciones NAT64. Los resultados de cada prueba (*throughput*, latencia o RTT) fueron sumados y puestos en la tabla 13. El ganador se distingue por estar en negritas. En el caso de TCP es el número más grande y en el caso de UDP e ICMP es el número más pequeño.

Aplicación	TCP	UDP	ICMP
Ecdysis	0	46.469	8.89
Lithuania	17015.796	56.514	8.411
TAYGA	17483.8	35.701	8.46
OpenBSD	17655.011	34.062	7.498
ITESM-NIC	17392.879	34.737	37.1786
ITESM-NIC-2	16996.2	33.649	8.4976

Tabla 13 Resultados de benchmarking

TCP (Ganador: OpenBSD)

Dentro de la traducción de TCP, OpenBSD obtuvo un *throughput* más alto que sus competidores, esto se puede entender porque el NAT64 de OpenBSD está integrado a la herramienta de filtrado de paquetes PF, que forma parte del sistema operativo. En segundo lugar se encuentra TAYGA, que aunque no es un sistema completamente con estados, se puede decir que los utiliza. Su desempeño se debe también al hecho de estar integrado con Netfilter, la herramienta para filtrar paquetes en Linux. ITESM-NIC llega en tercer lugar, Lithuania en cuarto. ITESM-NIC-2 en quinto lugar, lo que sugiere que la interfaz virtual en efecto disminuye el *throughput*. Ecdysis quedó en último lugar, como ya se había comentado, tuvo demasiados *kernel panics* durante las pruebas y se omitió su resultado porque no se pudieron obtener al menos 10 resultados consecutivos. Probablemente la razón de los *kernel panics* en el NAT64 de Ecdysis es el pobre uso de la estructura *Red-Black Tree* del *kernel*.

UDP (Ganador: ITESM-NIC-2)

La implementación que obtuvo un mejor *throughput* para la traducción del protocolo UDP fue ITESM-NIC-2, que utiliza la interfaz virtual para mandar y recibir paquetes. Esto permite inferir que el proceso de traducción de UDP no es tan demandante, y que tal vez la interfaz virtual no es la razón de que ITESM-NIC-2 obtuviera un mal *throughput* para TCP. En segundo lugar quedó OpenBSD, le siguen ITESM-NIC y TAYGA. Cabe recalcar que la diferencia entre estos últimos sistemas es mínima, por lo que la mayoría tienen buena latencia. Los que peor se desempeñaron fueron Ecdysis y Lithuania.

ICMP (Ganador: OpenBSD)

Dentro de la traducción del protocolo ICMP, OpenBSD obtuvo menor latencia que los demás sistemas, seguido de Lithuania, TAYGA, ITESM-NIC-2 y Ecdysis. ITESM-NIC fue el peor, se asume que es debido a que tiene dos problemas: pérdida de paquetes y/o latencia muy grande. Con esto se probó que la interfaz virtual ayuda a mejorar la latencia de este sistema en específico.

Capítulo 5 Conclusión

En resumen, dentro de este documento se pudo contestar el problema de investigación que se planteó en la introducción. No se encontró un modelo estándar para hacer pruebas de desempeño, por lo que en base en la literatura encontrada se propuso uno. Después se describió este modelo y se ejecutó. Se realizaron un serie de pruebas y experimentos y finalmente se recopilaron los resultados de estos. El modelo de benchmarking logra obtener buenos resultados para el *throughput* de http, latencia de DNS y RTT de ICMP. Pero el ambiente controlado y las pruebas definidas limitan el obtener un resultado más preciso para la utilización de CPU y de memoria. Por otro lado, se puede decir que el modelo logra identificar problemas en la implementación de algunos sistemas por ejemplo, Ecdysis e ITESM-NIC. El modelo también puede ser usado por desarrolladores a tomar una decisión informada sobre el diseño de su aplicación, como se puede ver en los resultados de ITESM-NIC e ITESM-NIC-2. Finalmente se puede decir que este modelo es el único que maneja los tres protocolos indicados por el RFC6146.

Con la obtención de los resultados se completa el análisis, con lo que se puede concluir que el desempeño de una aplicación NAT64, depende de los siguientes tres elementos:

1. Diseño, o arquitectura del sistema; por ejemplo, la decisión de cómo recibir/mandar paquetes, uso o no de una pool de IPv4.
2. Optimización de la programación. Por ejemplo, las implementación de OpenBSD y la implementación de NAT44 en Linux que usa TAYGA han sido optimizadas y trabajadas por años por desarrolladores del *kernel* profesionales. Una optimización en el código de ITESM-NIC podría ser trabajar solamente con apuntadores y solamente referenciarlos a través del resto del código.
3. Buena implementación y uso de las estructuras de datos. Un ejemplo es Ecdysis que al no usar de la mejor manera la estructura de datos *Red-Black Tree*, obtiene un pésimo rendimiento u ocasiona que el sistema se inhiba.

Trabajo futuro

Hablando sobre el benchmarking propuesto aquí, faltaría correr éstas pruebas en un ambiente de producción para que se puedan medir correctamente las métricas de utilización de memoria y CPU y también así tener datos mas exactos de Throughput y Latencia. Este benchmarking tal vez debería incluir solo a tres implementaciones: la nueva versión de ITESM-NIC que esta por salir a versión beta, OpenBSD y TAYGA, ya que la versión de Lithuania ya no se encuentra disponible en la web y Ecdysis ya no se esta actualizando constantemente.

Cabe mencionar que el monitoreo por medio de SNMP aportaría información más significativa en un ambiente de producción y no en un ambiente de pruebas controlado. Donde la herramienta Observium guarde información sobre el uso de CPU y memoria bajo un tráfico de usuarios reales, durante un periodo de tiempo mucho más largo. Sin embargo, se planteó la idea de instalar NAT64/DNS64 en el laboratorio de redes la escuela, este laboratorio tiene una red IPv6 y se le permitiría tener acceso a la red del campus, que todavía es en su totalidad IPv4.

En un ambiente de producción también se podría incrementar la duración de las pruebas para tener resultados más precisos, esto no fue posible en el ambiente de pruebas debido a la dificultad que implican las fallas en algunas de las implementaciones probadas.

Anexos

I. Levantar ambiente de pruebas (PASO 1)

Se adjunta a esta tesis un DVD que contiene 3 imágenes virtuales de los nodos dentro del ambiente de pruebas. Éstas imágenes contienen la configuración necesaria para cada aplicación a probar. También se incluye la imagen de OpenBSD, la cual se tiene que configurar para usar NAT64. Finalmente se incluye el código fuente para las diferentes versiones de NAT64 que fueron probadas. Algo importante a notar son las rutas en donde se encuentran las implementaciones de NAT64 dentro del nodo servidor, las cuales se muestran a continuación:

Directorio donde se encuentra Ecdysis:

/media/DVD/codigos/ecdysis-nf-nat64-20101117

Referirse a la sección II de los Anexos para instarlo correctamente. Asegúrese de tener el kernel 2.6.32-41-generic antes de usar o compilar el programa. Si se necesita compilar usar el comando make.

Directorio donde se encuentra linuxnat64:

/media/DVD/codigos/linuxnat64

Referirse a la sección II de los Anexos para instarlo correctamente. Asegúrese de tener el kernel 2.6.32-41-generic antes de usar o compilar el programa. Si se necesita compilar usar el comando make.

Directorio donde se encuentra ITESM-NIC:

/media/DVD/codigos/ITESM-NIC

Referirse a la sección II de los Anexos para instarlo correctamente. Asegúrese de tener el kernel 2.6.38-15-codigos antes de usar o compilar el programa. Si se necesita compilar usar el comando make dentro del folder mod. Para correr el programa se hace uso del script workflow.sh

Directorio donde se encuentra ITESM-NIC-2:

/media/DVD/codigos/Desktop/ITESM-NIC-2

Referirse a la sección II de los Anexos para instarlo correctamente. Asegúrese de tener el kernel 2.6.38-15-codigos antes de usar o compilar el programa. Si se necesita compilar usar el comando make dentro del folder mod. Para correr el programa se hace uso del script workflow.sh

Directorio donde se encuentra TAYGA:

/media/DVD/codigos/tayga-0.9.2

Para correr TAYGA solo se necesita usar el script star64.sh dentro de la máquina, referirse a la sección II de los Anexos para instrucciones detalladas.

OpenBSD

Dentro del DVD también se incluye la imagen para instalar OpenBSD 5.1 en una plataforma de 64 bits. Se puede descargar OpenBSD en la siguiente liga: <http://www.openbsd.org/ftp.html>. Esta imagen no está configurada como las otras 3 imágenes por lo que es necesario referirse a la sección II de los Anexos. La configuración es demasiado trivial como para incluir una imagen personalizada.

1) Topología del ambiente de pruebas

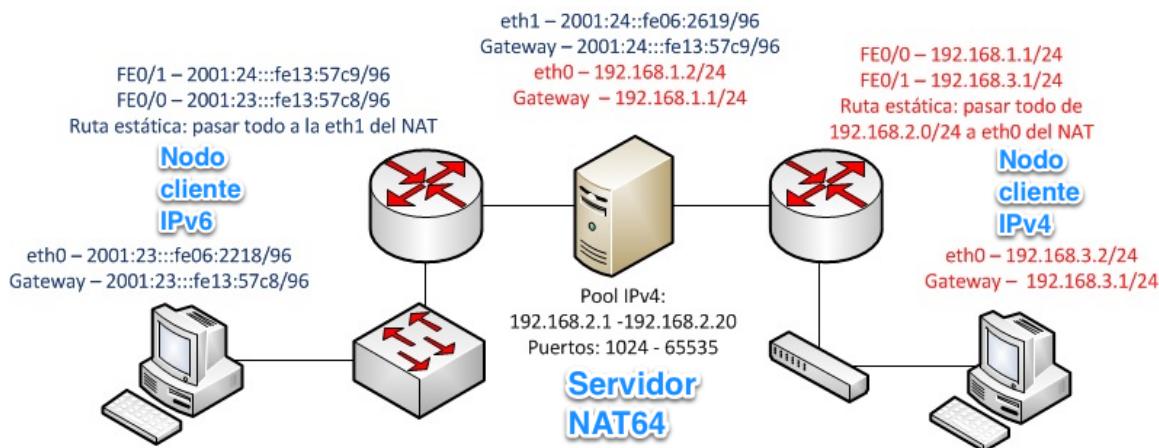


Figura 11 Topología del ambiente de pruebas

Referirse a la tabla 6 y 7 dentro de este documento para observar las especificaciones de las máquinas usadas. Cabe mencionar que para el nodo servidor NAT64 se usaron **dos máquinas diferentes**, a una se la instalado Linux y a otra se le instaló OpenBSD.

2) Configuración de red de nodo cliente IPv6

Editar el archivo /etc/network/interfaces, los permisos no se deben de cambiar.

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet6 static
    pre-up modprobe ipv6
    address 2001:23::fe06:2218
    netmask 96
    gateway 2001:23::fe13:57c8
    dns-nameservers ::1
    dns-search localhost
```

Después reiniciar el servicio, con el comando:

```
sudo /etc/init.d/networking restart
```

3) Configuración de red de nodo cliente IPv4

Editar el archivo /etc/network/interfaces, los permisos no se deben de cambiar.

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 192.168.3.2
    netmask 255.255.255.0
    network 192.168.3.0
    gateway 192.168.3.1
```

Después reiniciar el servicio, con el comando:

```
sudo /etc/init.d/networking restart
```

4) Configuración de red de nodo servidor Linux

Editar el archivo /etc/network/interfaces, los permisos no se deben de cambiar.

```
# This file describes the network interfaces available on
# your system
# and how to activate them. For more information, see
interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth1
iface eth1 inet6 static
    pre-up modprobe ipv6
    address 2001:24::fe06:2619
    netmask 96
    gateway 2001:24::fe13:57c9

auto eth0
iface eth0 inet static
```

```
address 192.168.1.2
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1
```

Después reiniciar el servicio, con el comando:

```
sudo /etc/init.d/networking restart
```

5) Configuración de red de nodo servidor OpenBSD

Editar el archivo /etc/hostname.nfe0, los permisos no se deben de cambiar.

```
inet6 2001:24::fe06:2619 96
!route -n add -inet6 default 2001:24::fe13:57c9
```

Editar el archivo /etc/hostname.rl0, los permisos no se deben de cambiar.

```
inet 192.168.1.2 255.255.255.0 192.168.1.255
!route add -net 192.168.3.0/24 192.168.1.1
```

NOTA: la terminación .nfe0 y .rl0 depende de la tarjeta de red instalada. Por lo que deberá ser diferente para una instalación nueva.

II. Configuración de NAT64 (PASO 2)

A continuación se detalla como instalar y configurar las diferentes versiones de NAT64. El código fuentes se encuentra en el DVD mencionado en el paso anterior, pero también se describe como descargar algunas versiones de Internet.

1) Descarga de *kernels* de Linux probados

Para las pruebas se usaron versiones de *kernel* de Linux específicos. Estos fueron descargados en la instalación de Ubuntu 10.04 del nodo servidor. Se usaron los siguientes comando para bajarlos:

```
sudo apt-get install linux-image-2.6.32-41-generic linux-headers-2.6.32-41-generic
sudo apt-get install linux-image-2.6.38-15-server linux-headers-2.6.38-15-server
sudo apt-get install build-essential
```

2) Configuración de Ecdysis

Ecdysis fue probado en Ubuntu 10.04 versión servidor, con *kernel 2.6.32-41-generic*. Referirse la sección II de los Anexos punto 1, para instalar los *kernels*. Se obtuvo el código fuente de Ecdysis de la siguiente liga, uno se tiene que registrar para bajarlo: <http://ecdysis.viagenie.ca/download/ecdysis-nf-nat64-20101117.tar.gz>

Una vez descargado el archivo, ejecutar los siguientes comandos:

```
tar xvzf ecdysis-nf-nat64-20101117.tar.bz2  
cd ecdysis-nf-nat64-20101117  
make
```

El siguiente archivo se llama *nat64-config.sh*, se encuentra dentro del directorio de Ecdysis. Este archivo se usa para configurar e instalar el módulo después de compilar.

```
#!/bin/bash  
  
#IPV4_ADDR="x.x.x.x"  
PREFIX_ADDR="64:ff9b::"  
PREFIX_LEN="96"  
  
echo *****  
echo "nf_nat64 setup"  
echo *****  
  
if [[ ! -z $1 ]]; then  
    IPV4_ADDR=$1;  
fi  
  
if [[ -z $IPV4_ADDR ]]; then  
    echo "Trying to guess the nat64 IPv4 address..."; echo  
    GUESS_IPV4_IF=$(ip route show | grep default | head -1 | sed -r 's/.+dev ([^ ]+) .+/\1/')  
    GUESS_PRIMARY_IPV4=$(ip addr show primary | grep "inet " | awk '{print $2}' | sed -r 's/[^\d]//g' | tail -1)  
  
    if [[ "$GUESS_PRIMARY_IPV4" == "" ]]; then  
        echo "Error: No IPv4 address found with default route found."  
        exit 1  
    fi  
  
    echo "Warning: Using a non-dedicated IPv4 address. NAT64 will partially  
work if you choose to continue with \"$GUESS_PRIMARY_IPV4\". See KNOWN_ISSUES  
for further information."  
    echo  
    echo "However, you can define a dedicated IPv4 address with:"  
    echo "# ifconfig eth0:1 w.x.y.z up"  
    echo "and run this script again:"  
    echo "# ./nat64-config.sh w.x.y.z"  
    echo "Ctrl-c to abort. Any other key will continue."  
    read i  
    IPV4_ADDR=$GUESS_PRIMARY_IPV4
```

```

fi

: ${IPV4_ADDR:?}

echo
echo "Info: Using $IPV4_ADDR as the NAT64 IPv4 address."
echo "Info: Using ${PREFIX_ADDR}/${PREFIX_LEN} as the NAT64 Prefix."

# Must have a global scope ipv6 address
if [[ $(ip -6 addr show scope global | grep inet6 | wc -l) == 0 ]]; then
    echo "No global scope ipv6 address!"
    exit 1
fi

echo
echo

set -ex

# Load the nf_nat64 module
#modprobe -r nf_nat64
insmod nf_nat64.ko nat64_ipv4_addr=$IPV4_ADDR nat64_prefix_addr=$PREFIX_ADDR
nat64_prefix_len=$PREFIX_LEN

# Enable the nat64 network interface
ifconfig nat64 up

# Install the route to the nat64 prefix
ip -6 route add ${PREFIX_ADDR}/${PREFIX_LEN} dev nat64

# Enable ipv6 and ipv4 forwarding
sysctl -w net.ipv4.conf.all.forwarding=1
sysctl -w net.ipv6.conf.all.forwarding=1

```

La imagen 11 se muestra como instalar Ecdysis usando el script que esta arriba. Donde el argumento a este script es la dirección: 192.168.1.2. La instalación requiere permisos de súper usuario, es decir, el comando *sudo* es necesario como se muestra en la imagen. Al teclear *sudo*, se le pide que ponga una clave de administrador.

Se tienen que modificar los permisos con el siguiente comando:

```
sudo chmod u+x nat64-config.sh
```

```

server@nat64:~/ecdysis-nf-nat64-20101117/ecdysis-nf-nat64-20101117$ sudo ./nat64-config.sh 192.168.1.2
*****
nf_nat64 setup
*****

Info: Using 192.168.1.2 as the NAT64 IPv4 address.
Info: Using 64:ff9b::/96 as the NAT64 Prefix.

+ insmod nf_nat64.ko nat64_ipv4_addr=192.168.1.2 nat64_prefix_addr=64:ff9b:: nat64_prefix_len=96
+ ifconfig nat64 up
+ ip -6 route add 64:ff9b::/96 dev nat64
+ sysctl -w net.ipv4.conf.all.forwarding=1
net.ipv4.conf.all.forwarding = 1
+ sysctl -w net.ipv6.conf.all.forwarding=1
net.ipv6.conf.all.forwarding = 1
server@nat64:~/ecdysis-nf-nat64-20101117/ecdysis-nf-nat64-20101117$ 

```

Figura 12 Instalación de Ecdysis

3) Configuración linuxnat64 de Lithuania

El linuxnat64 de Lithuania fue probado en Ubuntu 10.04 versión servidor, con *kernel* 2.6.32-41-generic. Referirse la sección II de los Anexos punto 1 para instalar los *kernel*s. El código fuente que obtenido de SourceForge pero actualmente ya no se encuentra disponible, para obtener el código fuente refiérase a la sección I de los Anexos para ver en donde se encuentra dentro del DVD proporcionado junto con ésta tesis.

El siguiente script se llama compile que se encuentra dentro de la carpeta de linuxnat64. Y se usa para compilar, configurar e instalar el módulo. Se muestra en negritas las líneas esenciales para que este módulo trabaje en el ambiente de pruebas.

```

#!/bin/sh

cd module
make

if [ $? -eq 0 ]
then
    cd ..
    echo "Copying nat64.ko..."
    cp module/nat64.ko .
    echo "Unloading old kernel module..."
    rmmod nat64.ko
    echo "Loading new kernel module..."
```

```

#insmod nat64.ko ipv4_address=193.219.61.75
prefix_address=2001:778:0:ffff:64::
    insmod nat64.ko ipv4_address=192.168.2.0/24 prefix_address=2001:24:1::
    echo "Bringing up nat64 device"
    ip link set nat64 up
    echo "Adding /96 route"
    #ip -6 route add default via 64:ff9b:: dev nat64
    ip route add 2001:24:1::/96 dev nat64 metric 1
    echo "Adding ipv4 route"
    #ip route add default via 192.168.2.0 dev nat64
    ip route add 192.168.2.0/24 dev nat64 metric 1
else
    cd ..
    echo "Error compiling nat64 module"
fi

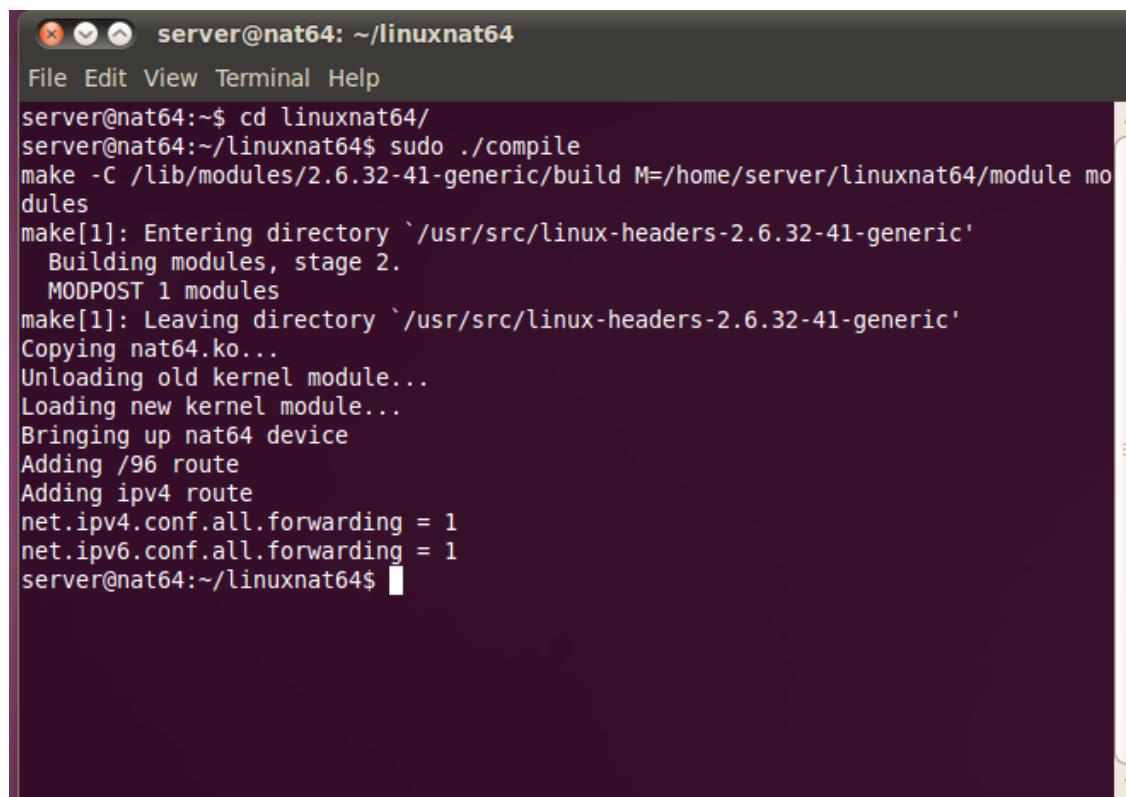
sysctl -w net.ipv4.conf.all.forwarding=1
sysctl -w net.ipv6.conf.all.forwarding=1

```

Se tienen que modificar los permisos con el siguiente comando:

```
sudo chmod u+x compile
```

La imagen 12 muestra su uso para instalar y configurar linuxnat64 de Lithuania.



```

server@nat64:~/linuxnat64
File Edit View Terminal Help
server@nat64:~$ cd linuxnat64/
server@nat64:~/linuxnat64$ sudo ./compile
make -C /lib/modules/2.6.32-41-generic/build M=/home/server/linuxnat64/module modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.32-41-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/usr/src/linux-headers-2.6.32-41-generic'
Copying nat64.ko...
Unloading old kernel module...
Loading new kernel module...
Bringing up nat64 device
Adding /96 route
Adding ipv4 route
net.ipv4.conf.all.forwarding = 1
net.ipv6.conf.all.forwarding = 1
server@nat64:~/linuxnat64$ 

```

Figura 13 Instalación de linuxnat64

4) Configuración de TAYGA

TAYGA fue probado en Ubuntu 10.04 versión servidor, con *kernel* 2.6.38-15-server. Referirse la sección II de los Anexos punto 1 para instalar los *kernels*. Para instalar TAYGA usar los siguientes comandos:

```
 wget http://www.litech.org/tayga/tayga-0.9.2.tar.bz2
 tar xvjf tayga-0.9.2.tar.bz2 && cd tayga-0.9.2
 ./configure
```

Este es el *output* del comando *./configure*:

```
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
configure: creating ./config.status
config.status: creating Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
```

Finalmente compilar con los siguientes commandos:

```
make && make install
mkdir -p /var/db/taiga
touch /usr/local/etc/tayga.conf
```

Editar archivo /usr/local/etc/tayga.conf

```
tun-device nat64
ipv4-addr 192.168.2.1
prefix 2001:24:1::/96
dynamic-pool 192.168.2.0/24
data-dir /var/db/tayga
```

Script start64.sh

Se tiene que crear el siguiente script para arrancar TAYGA

```
touch /home/server/start64.sh  
sudo chmod u+x start64.sh
```

Y poner el siguiente código:

```
#!/bin/bash  
  
sysctl -w net.ipv4.conf.all.forwarding=1  
sysctl -w net.ipv6.conf.all.forwarding=1  
tayga --mktun  
ip link set nat64 up  
ip addr add 192.168.1.2 dev nat64  
ip addr add 2001:24::fe13:57c9 dev nat64  
ip route add 192.168.2.0/24 dev nat64  
ip route add 2001:24:1::/96 dev nat64  
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE  
iptables -A FORWARD -i eth0 -o nat64 -m state --state RELATED,ESTABLISHED -j  
ACCEPT  
iptables -A FORWARD -i nat64 -o eth0 -j ACCEPT  
tayga
```

Una vez que se tengan estos archivos, ejecutarlos de la siguiente manera como se muestra en la figura 13:

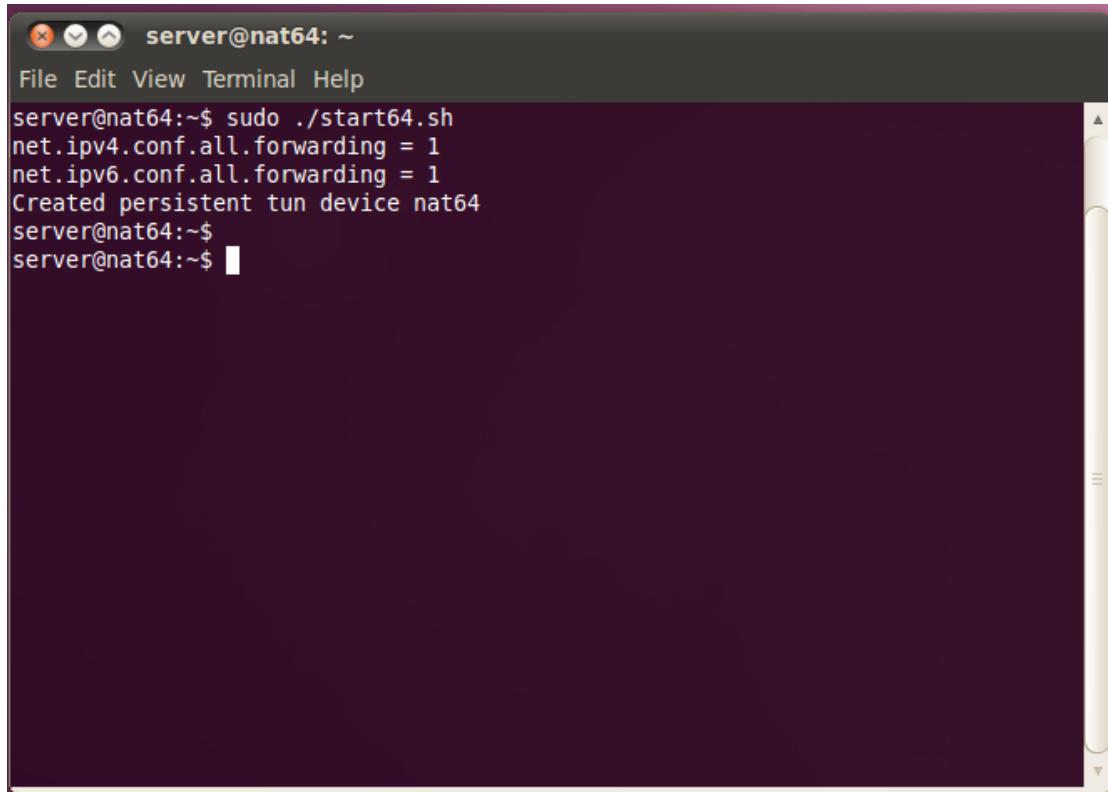


Figura 14 Instalación de TAYGA

5) Configuración de NAT64 en OpenBSD

Editar el archivo de configuración de *pf*, */etc/pf.conf* y agregar las siguientes líneas que se encuentran en negritas. Los permisos del archivo no se tienen que cambiar. Después de agregar las líneas mencionadas, se tiene que reiniciar el servidor.

```
#      $OpenBSD: pf.conf,v 1.50 2011/04/28 00:19:42 mikeb Exp $
#
# See pf.conf(5) for syntax and examples.
# Remember to set net.inet.ip.forwarding=1 and/or net.inet6.ip6.forwarding=1
# in /etc/sysctl.conf if packets are to be forwarded between interfaces.

set skip on lo

# filter rules and anchor for ftp-proxy(8)
#anchor "ftp-proxy/*"
#pass in quick inet proto tcp to port ftp divert-to 127.0.0.1 port 8021

# anchor for relayd(8)
#anchor "relayd/*"

pass      # to establish keep-state

pass in inet af-to inet6 from 64:ff9b::1
pass in inet6 af-to inet from 192.168.2.1

# rules for spamd(8)
#table <spamd-white> persist
#table <nospamd> persist file "/etc/mail/nospamd"
#pass in on egress proto tcp from any to any port smtp \
#      rdr-to 127.0.0.1 port spamd
#pass in on egress proto tcp from <nospamd> to any port smtp
#pass in log on egress proto tcp from <spamd-white> to any port smtp
#pass out log on egress proto tcp to any port smtp

#block in quick from urpf-failed to any# use with care

# By default, do not permit remote connections to X11
block in on ! lo0 proto tcp to port 6000:6010
```

6) Configuración de ITESM-NIC-2

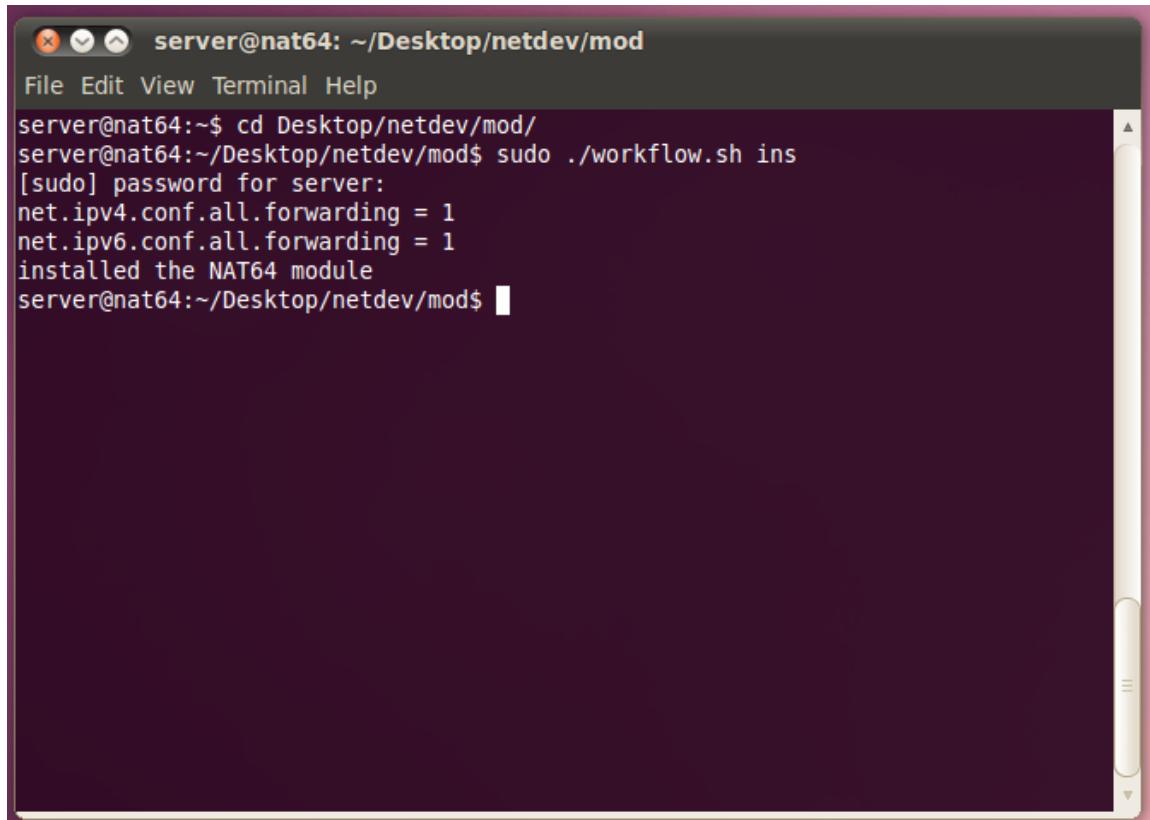
ITESM-NIC-2 fue probado en Ubuntu Linux 10.04 versión servidor, con *kernel* 2.6.38-15-server. Referirse la sección II de los Anexos punto 1 para instalar los *kernels*, en la sección I de los Anexos se indica la ruta en donde se encuentra este código fuente dentro del DVD que viene con este documento. También el código puede ser descargado con el siguiente comando desde cualquier ruta:

```
git clone https://github.com/magg/aporte1.git
```

Dentro de la carpeta mod se encuentra el archivo Makefile para compilar la aplicación con el comando: make. Y ahí mismo se encuentra el archivo /mod/workflow.sh el cual se tiene que modificar sus permisos. Todos los demás archivos pueden ser ignorados.

```
cd /home/server/aporte1/mod  
sudo chmod u+x workflow.sh
```

La siguiente imagen muestra los comandos necesarios para instalar:

A screenshot of a terminal window titled "server@nat64: ~/Desktop/netdev/mod". The window shows a command-line session where the user runs "cd Desktop/netdev/mod/" followed by "sudo ./workflow.sh ins". A password prompt "[sudo] password for server:" is shown, and the user enters their password. The output indicates that the "net.ipv4.conf.all.forwarding = 1" and "net.ipv6.conf.all.forwarding = 1" settings have been modified, and the module "NAT64" has been installed. The terminal window has a dark background and a light-colored text area. The title bar is dark with white text.

```
server@nat64:~$ cd Desktop/netdev/mod/  
server@nat64:~/Desktop/netdev/mod$ sudo ./workflow.sh ins  
[sudo] password for server:  
net.ipv4.conf.all.forwarding = 1  
net.ipv6.conf.all.forwarding = 1  
installed the NAT64 module  
server@nat64:~/Desktop/netdev/mod$
```

Figura 15 Instalación de ITESM-NIC-2

7) Configuración de ITESM-NIC

ITESM-NIC fue probado en Ubuntu 10.04 versión servidor, con *kernel* 2.6.38-15-server. Referirse la sección II de los Anexos punto 1 para instalar los *kernels*. La versión de ITESM-NIC probada se encuentra el DVD adjunto, referirse a la sección I de los Anexos para obtener el código fuente.

Instalar los siguientes paquetes:

```
sudo apt-get install iptables-dev, libssl-dev
```

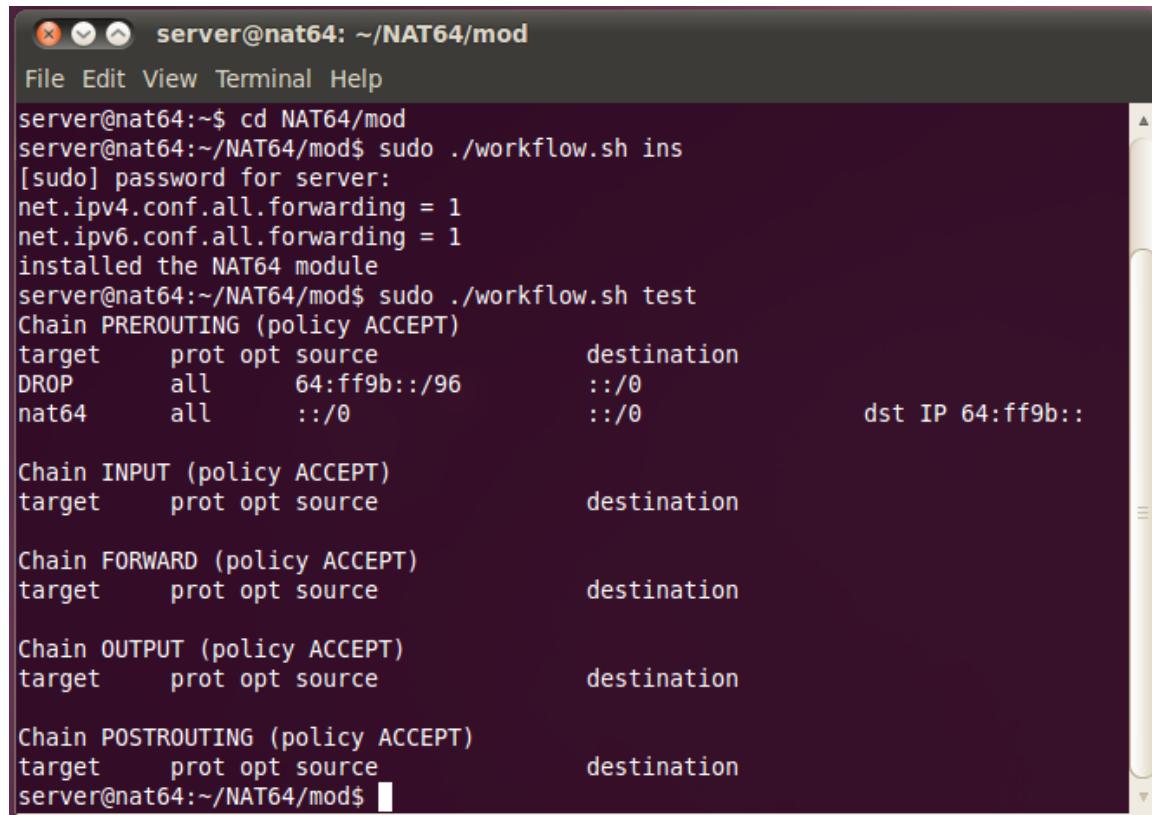
Dentro de la carpeta usr/ correr el siguiente comando:

```
make libxt_nat64.so  
sudo mv libxt_nat64.so /lib/iptables
```

Dentro de la carpeta mod se encuentra el archivo Makefile para compilar la aplicación con el comando: make. Y ahí mismo se encuentra el archivo /mod/workflow.sh el cual se tiene que modificar sus permisos.

```
cd NAT64/mod  
make  
sudo chmod u+x workflow.sh
```

La siguiente imagen muestra los comandos necesarios para instalar:



The screenshot shows a terminal window titled "server@nat64: ~/NAT64/mod". The user runs several commands to install the module and test its configuration. The output shows the module installed and the resulting iptables rules.

```
server@nat64:~$ cd NAT64/mod  
server@nat64:~/NAT64/mod$ sudo ./workflow.sh ins  
[sudo] password for server:  
net.ipv4.conf.all.forwarding = 1  
net.ipv6.conf.all.forwarding = 1  
installed the NAT64 module  
server@nat64:~/NAT64/mod$ sudo ./workflow.sh test  
Chain PREROUTING (policy ACCEPT)  
target    prot opt source          destination  
DROP      all    64:ff9b::/96    ::/0  
nat64     all    ::/0           ::/0          dst IP 64:ff9b::  
  
Chain INPUT (policy ACCEPT)  
target    prot opt source          destination  
  
Chain FORWARD (policy ACCEPT)  
target    prot opt source          destination  
  
Chain OUTPUT (policy ACCEPT)  
target    prot opt source          destination  
  
Chain POSTROUTING (policy ACCEPT)  
target    prot opt source          destination  
server@nat64:~/NAT64/mod$
```

Figura 16 Instalación de ITESM-NIC

III. Configuración de equipo de red (PASO 3)

A continuación se muestran los archivos de configuración para los enrutadores Cisco usados en el ambiente de pruebas. Se pone en negritas la configuración más importante para hacer funcionar este ambiente. Todo lo demás es configuración por defecto y no es necesaria.

a. Configuración de enrutador Cisco IPv4

```
Building configuration... Current configuration : 1041 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Router
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
!
resource policy
!
!
!
ip cef
!
! no ip domain lookup
!
!
!
voice-card 0
no dspfarm
!
!
!
!
!
!
!
!
interface FastEthernet0/0
ip address 192.168.1.1 255.255.255.0
duplex auto
speed auto
!
interface FastEthernet0/1
ip address 192.168.3.1 255.255.255.0
duplex auto
speed auto
!
interface Serial0/0/0
no ip address
shutdown
no fair-queue
clock rate 125000    !
interface Serial0/0/1
no ip address
shutdown
clock rate 125000
!
interface Serial0/1/0
no ip address
```

```

shutdown
clock rate 125000
!
interface Serial0/1/1
no ip address
shutdown
clock rate 125000
!
ip route 192.168.2.0 255.255.255.0 FastEthernet0/0 192.168.1.2
!
!
ip http server
no ip http secure-server
!
!
!
control-plane
!
!
!
!
line con 0
line aux 0
line vty 0 4
login
!
scheduler allocate 20000 1000 !
end

```

b. Configuración de enrutador Cisco IPv6

```

Current configuration : 1326 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname R1
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$2E/B$m3j1Kmt.OcJWhPR7qVj4c.
!
no aaa new-model
!
resource policy
!
!
!
ip cef
!
!
!
no ip domain lookup
!
```

```

ipv6 unicast-routing
ipv6 cef
!
!
voice-card 0
no dspfarm
!
!
!
!
interface FastEthernet0/0
no ip address
duplex auto
speed auto
ipv6 address 2001:23::/96 eui-64
ipv6 enable
!
interface FastEthernet0/1
no ip address
duplex auto
speed auto
ipv6 address 2001:24::/96 eui-64
!
!
interface Serial0/0/0
no ip address
shutdown
no fair-queue
clock rate 2000000
!
interface Serial0/0/1
no ip address
shutdown
clock rate 2000000
!
interface Serial0/1/0
no ip address
shutdown
clock rate 125000
!
interface Serial0/1/1
no ip address
shutdown
clock rate 125000
!
!
ip http server
no ip http secure-server
!
!
!
ipv6 route ::/0 FastEthernet0/1 2001:24::FE06:2619
!
!
!
!
```

control-plane

```
!
!
!
!
banner motd ^C
*****
!!!AUTHORIZED ACCESS ONLY!!!
WELCOME! TO CITY 17!
*****
^C
!
line con 0
 password cisco
 login
line aux 0
line vty 0 4
 password cisco
 login
!
scheduler allocate 20000 1000
!
end
```

IV. Configuración de software para pruebas (PASO 4)

Esta sección de los Anexos detalla como instalar y configurar el software necesario para realizar las pruebas descritas en la propuesta de benchmarking.

1) Configuración para NTP

Se instala con el comando:

```
sudo apt-get install -y ntp
```

Para establecer el servicio de NTP se usa el siguiente comando:

```
sudo sed -i 's/server ntp.ubuntu.com/server
ntp.ubuntu.com\nserver 127.127.1.0\nfudge 127.127.1.0 stratum
10/g' /etc/ntp.conf
sudo service ntp restart
```

2) Instalación de traceroute

Traceroute fue instalado en el nodo cliente IPv6, con el siguiente comando:

```
sudo apt-get install traceroute
```

3) Instalación de SSH

SSH fue instalado en los dos nodos cliente, IPv4 y el nodo IPv6, con el siguiente comando:

```
sudo apt-get install ssh
```

4) Instalación de Apache2

El servidor web Apache2 fue instalado en el Nodo Cliente IPv4, con el siguiente comando:

```
sudo apt-get install apache2
```

5) Instalación de Apache Bench

La herramienta Apache Bench fue instalada en el Nodo Cliente IPv6, con el siguiente comando:

```
sudo apt-get install apache2-utils
```

6) Configuración de servidor DNS

El servidor DNS fue instalado en el Nodo Cliente IPv4. Esta configuración se basó en el tutorial detallado en: <http://ubuntuforums.org/showthread.php?t=236093>.

Instalar BIND

```
sudo apt-get install bind9
```

Configurar los archivos de BIND

```
sudo vi /etc/bind/named.conf.local
```

Insertar lo siguiente

```
# This is the zone definition. replace example.com with your domain name
zone "example.com" {
    type master;
    file "/etc/bind/zones/example.com.db";
};

# This is the zone definition for reverse DNS. replace 0.168.192 with your network
address in reverse notation - e.g my network address is 192.168.0
zone "3.168.192.in-addr.arpa" {
```

```
type master;
file "/etc/bind/zones/rev.3.168.192.in-addr.arpa";
};
```

Crear las zonas

```
sudo mkdir /etc/bind/zones
sudo vi /etc/bind/zones/example.com.db
```

Insertar lo siguiente

```
// replace example.com with your domain name. do not forget the . after the domain
name!
// Also, replace ns1 with the name of your DNS server
example.com.    IN    SOA    ns1.example.com. admin.example.com. (
// Do not modify the following lines!
                                2006081401
                                28800
                                3600
                                604800
                                38400
)
// Replace the following line as necessary:
// ns1 = DNS Server name
// mta = mail server name
// example.com = domain name
example.com.    IN    NS        ns1.example.com.
example.com.    IN    MX    10    mta.example.com.

// Replace the IP address with the right IP addresses.
www          IN    A     192.168.3.2
mta          IN    A     192.168.3.2
ns1          IN    A     192.168.3.2
```

Crear la zona de *Reverse DNS*

```
sudo vi /etc/bind/zones/rev.3.168.192.in-addr.arpa
```

Insertar lo siguiente

```
//replace example.com with yoour domain name, ns1 with your DNS
server name.
// The number before IN PTR example.com is the machine address of
the DNS server. in my case, it's 1, as my IP address is
192.168.0.1.
@ IN SOA ns1.example.com. admin.example.com. (
                                2006081401;
                                28800;
```

```

        604800;
        604800;
        86400
)
1           IN      NS      ns1.example.com.
             IN      PTR     example.com

```

Reiniciar el servicio

```
sudo /etc/init.d/bind9 restart
```

Modificar la resolución de DNS

```
sudo vi /etc/resolv.conf
```

Insertar lo siguiente

```
// replace example.com with your domain name, and
192.168.0.1 with the address of your new DNS server.
search example.com
nameserver 192.168.3.2
```

7) Configuración de servidor DNSSEC

El servidor DNSSEC fue instalado en el Nodo Cliente IPv4. Esta configuración se basó en el tutorial detallado en: <http://www.howtoforge.com/configuring-dnssec-on-bind9-9.7.3-on-debian-squeeze-ubuntu-11.10>

En el servidor 1 (maestro) o Nodo Cliente IPv4 crear el archivo /etc/bind/pri.example.org

```
$TTL      3600
@       IN      SOA      server1.example.com.
zonemaster.example.com. (
                           2012041305      ; serial, todays date +
todays serial #
                           7200      ; refresh, seconds
                           540       ; retry, seconds
                           604800    ; expire, seconds
                           86400 )   ; minimum, seconds
;

example.org. 3600 A      1.2.3.4
example.org. 3600      MX      10      mail.example.org.
example.org. 86400     NS      server1.example.com.
example.org. 86400     NS      server2.example.com.
example.org. 3600      TXT     "v=spf1 a mx ptr -all"
mail 3600 A            1.2.3.4
www 3600 A             1.2.3.4
```

Editar el archivo named.conf.local

```
zone "example.org" {
    type master;
    allow-transfer {192.168.3.3;};
    also-notify {192.168.3.3;};
    file "/etc/bind/pri.example.org.signed";
};
```

Instalar los siguientes paquetes

```
sudo apt-get install dnssec-tools libnet-dns-sec-perl libmailtools-perl
libcrypt-openssl-random-perl
```

Editar named.conf.options

```
options {
    directory "/var/cache/bind";

    auth-nxdomain no;      # conform to RFC1035
    listen-on-v6 { any; };
    dnssec-enable yes;
    dnssec-validation yes;
    dnssec-lookaside auto;
    //bindkeys-file "/etc/bind/bind.keys";
};

include "/etc/bind/bind.keys";
```

Reiniciar BIND

```
/etc/init.d/bind9 restart
```

Firmar la zona con el comando:

```
zonesigner -genkeys -usensec3 -zone example.org pri.example.org
```

En otra computadora, nodo cliente 2 IPv4, editar el archivo /etc/bind/named.conf.local

```
zone "example.org" {
    type slave;
    masters {192.168.3.2;};
    allow-notify {192.168.3.2;};
    allow-transfer {none;};
    file "/etc/bind/slave/sec.example.org.signed";};
```

Editar named.conf.options

```

options {
    directory "/var/cache/bind";
    auth-nxdomain no;      # conform to RFC1035
    listen-on-v6 { any; };
    dnssec-enable yes;
    dnssec-validation yes;
    dnssec-lookaside auto;
    //bindkeys-file "/etc/bind/bind.keys";
};

include "/etc/bind/bind.keys";

```

Reiniciar BIND

```
/etc/init.d/bind9 restart
```

Ahora en un tercer nodo, nodo cliente 3 IPv4 editar named.conf.options, la sección de managed-keys se toma del archivo .KSK dentro del nodo cliente IPv4.

```

options {
    directory "/var/cache/bind";

    forwarders {
        192.168.3.2; 192.168.3.3;
    };

    auth-nxdomain no;      # conform to RFC1035
    listen-on-v6 { any; };
    dnssec-enable yes;
    dnssec-validation yes;
    dnssec-lookaside auto;
};

include "/etc/bind/bind.keys";

managed-keys {
    example.org. initial-key 257 3 8
    "AwEAAbjthg82WErIMm+gcsOeNlI6j7/9VuihQtYVnt9dOFWeddfZxlbv
    VIFKk1xBLMmBt4Z5GULTDKg+2BA6hGq3UGTHJMg1cpYTZtUBF4R1LnxE
    2KB15rBFtU8b3C80trpGsEI/VUWei5IPopFU04QMDCQkXBiulwHbG6Z
    cynlvYeaUC94CVabjTPpO95BysAZqBrxQsWyokMWwMtX6V0+uYlzGIU2
    OJazpYkWsIrAfpY2dRL15pugx4gCWMZwdsrfiHZSS7n1DCaDbAgsTS5t
    QiU4zy2YQ7vst7U4Zmh0+WbfHefeyVByCdiQaF2UmVsnnTxuEtu1Y3SS
    ClmDzq2/wW8=";
};

```

Editar /etc/resolv.conf

```
nameserver 127.0.0.1
```

Reiniciar BIND

```
/etc/init.d/bind9 restart
```

8) Instalación de DNSperf

La herramienta DNSperf fue instalada en el Nodo Cliente IPv6, con los siguiente comandos:

```
sudo apt-get install libbind-dev build-essential libssl-dev
sudo apt-get install dnsutils bind9
sudo apt-get install libcap-dev tshark
sudo apt-get install libxml2-dev
wget ftp://ftp.isc.org/isc/bind9/9.9.0/bind-9.9.0.tar.gz
tar zxvf bind-9.9.0.tar.gz
sudo cp -p bind-9.9.0/lib/isc/include/isc/hmacsha.h /usr/include/isc/
sudo ln -s /usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2.2
/usr/lib/x86_64-linux-gnu/libgssapi_krb5.so
wget ftp://ftp.nominum.com/pub/nominum/dnsperf/2.0.0.0/dnsperf-src-
2.0.0.0-1.tar.gz
tar zxvf dnsperf-src-2.0.0.0-1.tar.gz
cd dnsperf-src-2.0.0.0-1/
./configure
make
sudo make install
```

9) Instalación de Observium

La herramienta Observium fue instalada en el Nodo Cliente IPv6, con los siguiente comandos:

Instalar los siguientes paquetes:

```
sudo aptitude install libapache2-mod-php5 php5-cli php5-mysql php5-gd
php5-snmp php-pear snmp graphviz subversion mysql-server mysql-client
rrdtool \
fping imagemagick whois mtr-tiny nmap ipmitool
```

Instalar las siguientes librerías PEAR

```
pear install Net_IPv6
pear install Net_IPv4
mkdir -p /opt/observium && cd /opt
svn co http://www.observium.org/svn/observer/trunk observium
cd observium
cp config.php.default config.php
```

Crear la base de datos

```
mysql -u root -p <mysql root password>
mysql> CREATE DATABASE observium;
```

```
mysql> GRANT ALL PRIVILEGES ON observium.* TO 'observium'@'localhost'  
-> IDENTIFIED BY '<observium db password>';
```

Editar config.php para reflejar los datos de base datos

Crear el esquema:

```
php includes/sql-schema/update.php
```

Crear directorios:

```
mkdir graphs rrd  
chown www-data.www-data graphs rrd
```

Cambiar el contenido de /etc/apache2/sites-available/default a:

```
<VirtualHost *:80>  
    ServerAdmin webmaster@localhost  
    DocumentRoot /opt/observium/html  
    <Directory />  
        Options FollowSymLinks  
        AllowOverride None  
    </Directory>  
    <Directory /opt/observium/html/>  
        Options Indexes FollowSymLinks MultiViews  
        AllowOverride All  
        Order allow,deny  
        allow from all  
    </Directory>  
    ErrorLog ${APACHE_LOG_DIR}/error.log  
    LogLevel warn  
    CustomLog ${APACHE_LOG_DIR}/access.log combined  
    ServerSignature On  
</VirtualHost>
```

Usar los siguientes comandos

```
a2enmod rewrite  
apache2ctl restart
```

Crear un usuario, como usuario yo puse “magg”, como password “12345” y nivel “10”

```
cd /opt/observium  
. ./adduser.php <username> <password> <level>
```

Añadir un host para monitorear, como hostname yo puse “nat64”, community “natserver”

```
./addhost.php <hostname> <community> v2c
```

Correr estos comandos

```
./discovery.php -h all  
./poller.php -h all
```

Editar el archivo /etc/cron.d/observium

```
33 */6 * * * root /opt/observium/discovery.php -h all >> /dev/null 2>&1  
*/5 * * * * root /opt/observium/discovery.php -h new >> /dev/null 2>&1  
*/5 * * * * root /opt/observium/poller.php -h all >> /dev/null 2>&1
```

Ahora se necesita instalar un cliente SNMP en el nodo Servidor NAT64 de Linux:

```
sudo apt-get install snmpd
```

Editar la variable SNMPOPTS en /etc/default/snmpd para que se vea así:

```
SNMPOPTS=' -Lsd -Lf /dev/null -u snmp -p /var/run/snmpd.pid'
```

El archivo /etc/snmp/snmpd.conf se debe ver de esta manera:

```
agentaddress udp:161  
agentaddress udp6:161  
  
rocommunity6 natserver  
com2sec6 readonly default natserver  
group MyROGroup v1 readonly  
group MyROGroup v2c readonly  
group MyROGroup usm readonly  
  
view all included .1 80  
access MyROGroup "" any noauth exact all none none  
  
syslocation CETEC Torre Norte CT-627  
syscontact Miguel <maggonzz@gmail.com>  
pass .1.3.6.1.2.1.31.1.1.18 /usr/local/lib/ifAlias  
extend .1.3.6.1.4.1.2021.7890.1 distro /usr/bin/distro
```

Descargar lo siguiente

```
wget http://www.observium.org/svn/observer/trunk/scripts/distro  
mv distro /usr/bin/distro  
chmod 755 /usr/bin/distro  
sudo /etc/init.d/snmpd restart
```

V. Ejecución del benchmarking (PASO 5)

En esta sección de los Anexos se presentan los comandos para cada prueba del benchmarking:

1) Comandos para la prueba de funcionalidad

Para la prueba de SSH se usó:

```
ssh cliente2@64:ff9b::192.168.3.2
```

Para la prueba de traceroute6 se usó el siguiente comando:

```
traceroute -6 -l 64:ff9b::192.168.3.2
```

Finalmente para la prueba de NTP se corrió el siguiente comando:

```
ntpdate 64:ff9b:192.168.3.2
```

2) Comandos para la prueba de desempeño.

- **Prueba de paquetes pequeños**

- c. TCP

```
ab -n 100 -c 10 http://[64:ff9b::192.168.3.2]/  
ab -n 200 -c 20 http://[64:ff9b::192.168.3.2]/  
ab -n 400 -c 40 http://[64:ff9b::192.168.3.2]/
```

- d. UDP

En este caso *dns-test* es un archivo que contiene la URL example.com en cada línea. El archivo debe contener 100 líneas para la primera prueba, 200 para la segunda y 400 para la tercera prueba. Por ejemplo:

```
example.com A
```

```
dnsperf -f inet6 -d dns-test -s 64:ff9b::192.168.3.2
```

- e. ICMP

```
ping6 -c 10 64:ff9b::192.168.3.2  
ping6 -c 20 64:ff9b::192.168.3.2  
ping6 -c 40 64:ff9b::192.168.3.2
```

- **Prueba de paquetes grandes**

a. TCP

```
ab -n 100 -c 10 http://[64:ff9b::192.168.3.2]/test.html  
ab -n 200 -c 20 http://[64:ff9b::192.168.3.2]/test.html  
ab -n 400 -c 40 http://[64:ff9b::192.168.3.2]/test.html
```

b. UDP

En este caso *dns-test* es un archivo que contiene la URL `example.org` en cada línea. El archivo debe contener 100 líneas para la primera prueba, 200 para la segunda y 400 para la tercera prueba. Por ejemplo:

```
example.org A
```

```
dnsperf -f inet6 -d dns-test -s 64:ff9b::192.168.3.2 -e -D
```

c. ICMP

```
ping6 -c 10 64:ff9b::192.168.3.2 -s 1200  
ping6 -c 20 64:ff9b::192.168.3.2 -s 1200  
ping6 -c 40 64:ff9b::192.168.3.2 -s 1200
```

3) Ejemplo de resultados de confiabilidad

Estas tablas contienen los resultados al usar la ecuación 4, la fórmula de confianza. Se muestran dos casos: paquetes pequeños y paquetes grandes para TCP.

Aplicación	Ronda 1	Ronda 2	Ronda 3
Ecdysis	N/A	N/A	N/A
Lithuania	0.1051	0.0851	0.0978
TAYGA	0.0539	0.0678	0.0894
OpenBSD	0.0269	0.0628	0.1373
ITESM-NIC	0.0723	0.0688	0.0781
ITESM-NIC-2	0.0569	0.0477	0.0336

Tabla 14 Resultados de t-student de paquetes pequeños TCP

Aplicación	Ronda 1	Ronda 2	Ronda 3
Ecdysis	N/A	N/A	N/A
Lithuania	0.1481	0.0237	0.0160
TAYGA	0.0629	0.0241	0.0147
OpenBSD	0.0384	0.0311	0.0901
ITESM-NIC	0.0810	0.0272	0.0157
ITESM-NIC-2	0.0361	0.0507	0.0191

Tabla 15 Resultados de t-student de paquetes grandes TCP

Bibliografía

- Bagnulo, M., Matthews, P., & van Beijnum, I. (2011). Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers (Vol. Request for Comments: 6146): Internet Engineering Task Force (IETF).
- Bajpai, V., Melnikov, N., Sehgal, A., & Schönwälter, J. (2012). *Flow-Based Identification of Failures Caused by IPv6 Transition Mechanisms*. Paper presented at the 6th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2012). Retrieved from <http://www.iurs.org/thesis/72790138.pdf>
- Baker, F., Li, X., Bao, C., & Yin, K. (2011). Framework for IPv4/IPv6 Translation (Vol. Request for Comments: 6144): Internet Engineering Task Force (IETF).
- Dan, W. (2010). Network Address Translation: Extending the Internet Address Space. *Internet Computing, IEEE*, 14(4), 66-70.
- Deng, X., Wang, L., Zheng, T., Gu, D., & Burgey, E. (2011). *Analysis on IPv6 Transition Solutions and Service Tests*. Paper presented at the ICNS 2011 : The Seventh International Conference on Networking and Services. Retrieved from http://www.thinkmind.org/download.php?articleid=icns_2011_4_40_10185
- Dionne, J.-P., Perreault, S., & Blanchet, M. (2010). *Ecdysis: Open-Source DNS64 and NAT64*. Paper presented at the AsiaBSDCon. Retrieved from <http://www.viagenie.ca/publications/2010-03-13-asiabsdcon-nat64.pdf>
- Geer, D. (2008). Technology News. *Computer*, 41(10), 16-19.
- Hassan, M., & Jain, R. (2004). *High performance TCP/IP networking: concepts, issues, and solutions*: Pearson/Prentice Hall.
- Hazeyama, H., Yamagishi, Y., Ueno, Y., Yokoishi, T., Sato, H., & Ishibashi, H. (2011). *How much can we survive on an IPv6 network?: experience on the IPv6 only connectivity with NAT64/DNS64 at WIDE camp 2011 Autumn*. Paper presented at the Proceedings of the 7th Asian Internet Engineering Conference.
- ITESM, C. M. (2011a). *Implementación NAT64/DNS64 Reporte de Investigación*. Monterrey, Mexico: Monterrey Institute of Technology and Higher Education.
- ITESM, C. M. (2011b). *Implementación NAT64/DNS64 Reporte Técnico*. Monterrey, Mexico: Monterrey Institute of Technology and Higher Education.
- Kriukas, J. (2010). *NAT64 Protocol Implementation*. Kaunas University of Technology, Kaunas, Lithuania.
- Llanto, K., & Yu, W. (2012). *Performance of NAT64 versus NAT44 in the Context of IPv6 Migration*. Paper presented at the Proceedings of The International MultiConference of Engineers and Computer Scientists 2012 (IMECS 2012). Retrieved from http://www.iaeng.org/publication/IMECS2012/IMECS2012_pp638-645.pdf
- Lähteenmäki, E. (2011). *Testing and Evaluation of a DNS64/NAT64 System*. Tampere University of Techonology, Tampere, Finland.
- Mauerer, W. (2008). *Professional Linux Kernel Architecture*: John Wiley & Sons.
- Naito, K., Mori, K., & Kobayashi, H. (2012). *Kernel Module Implementation of IPv4/IPv6 Translation Mechanisms for IPv4-oriented Applications*. Paper presented at the ICN 2012, The Eleventh International Conference on Networks. Retrieved from http://www.thinkmind.org/download.php?articleid=icn_2012_10_10_10126
- Rondou, P. (2011). *Address family translation (IPv6/IPv4) in a Linux kernel*. University of Liège, Liège, Belgium.

- Skoberne, N., & Ciglaric, M. (2011). Practical Evaluation of Stateful NAT64/DNS64 Translation. *Advances in Electrical and Computer Engineering*, 11(3), 49-54.
- Yu, S.-y., & Carpenter, B. (2012). *Measuring IPv4-IPv6 translation techniques*. New Zealand: University of Auckland Computer Science Department.