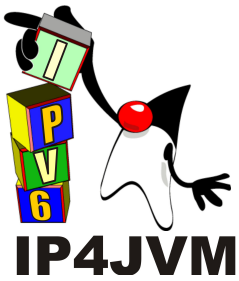

Instituto de Computación

Facultad de Ingeniería

Universidad de la República

Proyecto de Grado

CERTIFICACION DE IP4JVM

Diciembre de 2011

Autor: Daniel Enrique Rosano Lorenzo

Tutores
Ariel Sabiguero Yawelak
Leandro Scasso

Tribunal
Cecilia Apa
Javier Baliosian
Lorena Etcheverry

Resumen

El presente informe describe el proceso de preparación para la certificación por medio de una plataforma desarrollada por terceros, de la herramienta IP4JVM.

IP4JVM consiste en un framework enteramente programado en lenguaje Java que implementa un stack de protocolos de red, particularmente IPv6. Este proyecto viene siendo desarrollado desde el año 2006 y se encuentra ya en su cuarta generación, donde se apunta a tener un producto que cumpla las normas de IPv6 Ready Logo. Para ello se necesitan herramientas que permiten realizar las pruebas.

TAHI es una de las organizaciones que se dedica a desarrollar tecnologías para la verificación para IPv6 y esas tecnologías son la que se usan en el proyecto.

IRISA es un instituto francés que se dedica a la investigación en informática, y en particular comenzaron el desarrollo de la herramienta de interoperabilidad que luego fuera mejorada por Ariel Sabiguero, y esta fue utilizada en este proyecto para realizar el testing de interoperabilidad.

El objetivo del proyecto es hacer que la implementación existente pase los tests de conformidad e interoperabilidad del v6RL (v6 Ready Logo).

A lo largo del trabajo se distinguen 4 fases, a saber: el estudio previo, la instalación del ambiente de testing, la ejecución del conjunto de pruebas desarrolladas por TAHI e IRISA junto a la corrección de los errores hallados, y por último la documentación del proceso.

Palabras Claves

Networking en Java. Test de conformidad. Test de interoperabilidad. IPv6.

Índice

1 Introducción

El presente documento contiene la descripción general del proyecto de grado realizado por Daniel Rosano, el cual consiste en la preparación para la certificación del proyecto IP4JVM. Hablamos de preparación ya que este proyecto no consiste en la certificación real, porque ello implicaría viajar a un laboratorio certificado en Japón o Francia con nuestra máquina con IP4JM.

IP4JM es una implementación en Java de un stack de protocolos, de forma de proveer a la máquina virtual de Java la capacidad de manejar esta funcionalidad no incluida en la misma. El conjunto de protocolos en el que se basa el proyecto es la suite IPv6 el cual se creó con la idea de sustituir IPv4 (actual versión usada del protocolo de Internet) para solucionar varios problemas que éste presenta.

En su primera fase fue desarrollado por Laura Rodríguez a modo de pasantía entre marzo y noviembre del año 2006, continuado por Roger Abelenda e Ignacio Corrales [7] como parte de su Proyecto de Grado en el año 2007, y complementado por Leandro Scasso y Marcos Techera en el año 2008 [8]. Además en 2011 se desarrollaron las capacidades de MIPv6 por parte de Martín Giupponi, Ignacio Comesaña e Ignacio Lussich.

Los tutores del proyecto son Ariel Sabiguero, quien es tutor desde el comienzo del proyecto IP4JVM y Leandro Scasso, quien fue parte de la tercera generación del desarrollo de IP4JVM. Toda la documentación del proyecto, puede ser encontrada en el sitio personal [26] del tutor del proyecto.

El contexto de este trabajo es el proyecto de grado de la carrera Ingeniería de Computación y como tal cuenta con la intervención del Instituto de Computación de la Facultad de Ingeniería al cual pertenecen los tutores.

1.1 Motivación

Vista la gran madurez de IP4JVM (con más de 4 años de desarrollo continuo) se decidió que era un buen momento para certificar el producto. Por ello, en esta etapa de IP4JVM, la idea es certificarlo contra las herramientas de testing semiautomáticas provistas por TAHI [2] e IRISA, de manera de confirmar que IP4JVM cumple con los estándares definidos en los siguientes RFC y por lo tanto demostrando su calidad y valor como herramienta didáctica.

La suite de protocolos IPv6 que se cubren en los tests son los especificados en:

- RFC 2460 [11]: Internet Protocol, Version 6 (IPv6) Specification
- RFC 4861 [13]: Neighbor Discovery for IP version 6 (IPv6)
- RFC 4862 [14]: IPv6 Stateless Address Autoconfiguration
- RFC 1981 [10]: Path MTU Discovery for IP version 6
- RFC 4443 [12]: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification

Considerando además la siguiente actualización:

- RFC 5095 [25]: Deprecation of Type 0 Routing Headers in IPv6 (Updates 2460)

1.2 Objetivos

El objetivo último del proyecto es tener un stack de protocolos programado en Java, que cumpla los estándares definidos por el IPv6RL y supere los tests de conformidad e interoperabilidad

Los objetivos particulares del proyecto son:

- Familiarizarse con los protocolos implementados por IP4JVM y su correspondiente implementación.
- Crear el ambiente de testing, el cual consiste de un conjunto de máquinas virtuales (VM). Esto incluye la instalación de los dos sistemas operativos que se usan: FreeBSD y OpenSUSE [9].
- Instalar IP4JVM en la VM con OpenSUSE.
- Instalar las herramientas de testing en la VM con FreeBSD.
- Realizar corridas de testing y las necesarias modificaciones al código fuente de IP4JVM, de manera que todos los tests (conformidad e Interoperabilidad) resulten exitosos.
- Documentar el proceso.

1.3 Organización del documento

En la sección 2 se describe el estado del arte. Esto implica el estado inicial de la herramienta IP4JVM, de las herramientas auxiliares que se utilizaron para la ejecución del proyecto y las organizaciones que proveen dichas herramientas.

En la sección 3 y 4 se muestra el desarrollo del proceso de testing de conformidad e interoperabilidad respectivamente, describiendo sus principales fases y objetivos individuales. Se describen las distintas categorías de pruebas que se llevan a cabo, el estado inicial de las pruebas y su evolución.

En la sección 5 se presentan las conclusiones que fueron extraídas del proyecto y el trabajo a futuro que debería realizarse. Luego, en la sección 6 se presenta un pequeño glosario con los términos más utilizados en este documento. Por último, en la sección 7 se detallan las referencias utilizadas.

Además existen un conjunto de Anexos que detallan ciertas partes del proyecto a más nivel. En el Anexo I se proporciona un manual de instalación de FreeBSD, muy útil para quienes tengan que instalar este sistema operativo por primera vez.

En el Anexo II se detalla toda la instalación y configuración necesaria para utilizar la herramienta v6eval [15] mientras que en el Anexo III se describe todo el proceso de instalación de la herramienta IP4JVM a un sistema operativo OpenSuse.

Por último en los Anexos IV y V se describe en detalle causas y las soluciones de los diferentes problemas que surgieron en los tests de conformidad e interoperabilidad respectivamente.

1.4 Público Objetivo

Este documento está orientado a personas del perfil informático con cierto conocimiento de redes de computadoras y su organización jerárquica de capas; en particular es deseable que el lector tenga conocimiento del funcionamiento del stack IPv6. También es importante que el lector tenga los conocimientos básicos de testing y algún conocimiento de la comunidad IPv6, aunque no es excluyente.

2 Estado del arte

En esta sección se brindarán datos acerca de los proyectos anteriores y el estado en el que se encuentra. Esta información no es irrelevante, ya que el trabajo realizado anteriormente es el punto de partida de este proyecto, y por lo tanto, el estado de la herramienta y su evolución marcarán las pautas del trabajo y objetivos.

2.1 Datos generales de proyectos anteriores

Como se mencionó brevemente en la introducción, este proyecto se da como la continuación de un proyecto comenzado en 2006. Dicho trabajo fue desarrollado por Laura Rodríguez, enmarcado en el contexto de una pasantía realizada en Francia en el instituto IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires) que a su vez forma parte del la INRIA (Institut National de Recherche en Informatique et en Automatique), instituto francés que se dedica a la investigación dentro de las ciencias de la computación.

Laura efectuó su pasantía entre marzo y noviembre del año 2006, logrando en este tiempo los objetivos que se plantearon en sus inicios. Estos objetivos consistían, entre otros, en tener un framework programado en Java mediante el cual se puede implementar un stack de protocolos y agregar distintas implementaciones de protocolos que se desee, éstos también programados en Java.

Además de este framework se implementó parte de los protocolos Ethernet II, IPv6 y ICMPv6, con lo cual se podía comprobar el correcto funcionamiento mediante el uso del comando ping, el cual está especificado en el RFC 4443 referente al protocolo ICMPv6.

También llegó a implementarse algo de UDP y fue probado a través de una aplicación sencilla que consistía en el envío de mensajes desde un nodo a otro.

Es necesario, además, resaltar que los tutores de este proyecto fueron Ariel Sabiguero (quién a su vez es tutor de este proyecto), perteneciente al Instituto de Computación de la Facultad de Ingeniería (InCo) y Cesar Viho, quien pertenece al INRIA.

A continuación, en el año 2007 Roger Abelenda e Ignacio Corrales complementaron las implementaciones de Ethernet II, IPv6, ICMPv6 y UDP; además de realizar la implementación del protocolo TCP [RFC793] y conseguir que Tomcat pudiera ser utilizado junto a IP4JVM.

En el año 2008, Leandro Scasso y Marcos Techera, dedicaron su proyecto a complementar el trabajo de las anteriores etapas de IP4JVM. Implementaron el protocolo DHCPv6, las funcionalidades mínimas de un router IPv6 y una aplicación web que permite configurar de manera más amigable el proyecto.

En 2011, Martín Giupponi, Ignacio Comesaña e Ignacio Lussich; un grupo de estudiantes de ingeniería eléctrica, implementaron el protocolo MIPv6 (Mobile IPv6) lo

que permite que IP4JVM pueda ser utilizado en dispositivos móviles como celulares, laptops o PDA's.

2.2 IPv6 Ready Logo Committee

IPv6 Ready Logo Committee, es una organización destinada a definir las especificaciones de los tests de conformidad e interoperabilidad que permitan obtener el IPv6 Ready Logo.

Entre los objetivos del IPv6RL están:

- Verificar la implementación de protocolos y validar la interoperabilidad de los productos IPv6.
- Proveer acceso a herramientas de testing libres.
-

Proveer laboratorios de testing IPv6 Ready Logo a través del mundo, dedicados a brindar asistencia

2.3 IPv6 Ready Logo

El IPv6 Ready Logo [1] es una certificación que se divide en dos fases

- Fase 1: Logo de plata (170 tests de conformidad)
- Fase 2: Logo de oro (376 tests de conformidad + 36 tests de interoperabilidad)

Los tests de la Fase 1, refieren a un conjunto básico de test de conformidad que tiene que cumplir un dispositivo que pretenda brindar servicios de IPv6. La Fase 2 comprende un conjunto más amplio de tests de conformidad y además verifica la interoperabilidad entre dispositivos IPv6 a través de un conjunto de tests específicos.
En este proyecto de grado se apunta a obtener la certificación IPv6RL Fase 2.

Sin embargo, a pesar de contar con los casos de prueba definidos; realizar el testing de una aplicación de red manualmente, no es nada sencillo. Las múltiples condiciones que describen los diferentes RFC, las múltiples salidas, la posibilidad de perder paquetes en la red, hacen que la tarea de testing sea casi imposible de realizar manualmente. Por ello existen organizaciones que generan herramientas para realizar los tests.

2.4 TAHI

El proyecto TAHI [2] fue formado en 1998 con el objetivo de desarrollar y proporcionar tecnologías de verificación para IPv6.

El proceso de crecimiento de IPv4 fue la historia del encuentro con varios tipos de

obstáculos y vencimiento de esos obstáculos. Para evitar la repetición de este tipo de desarrollo, resultó imprescindible la creación de tecnologías de verificación para IPv6.

Para esto TAHI:

1. Investiga y desarrolla pruebas de conformidad y pruebas de interoperabilidad para IPv6.
2. Ayuda a las actividades de la parte de calidad ofreciendo tecnologías de verificación que se desarrollan en el proyecto TAHI para mejorar la eficiencia del desarrollo.
3. Ofrece los resultados y los frutos del proyecto al público de forma gratuita. Cualquier desarrollador de IPv6 puede utilizar los resultados y los frutos del proyecto TAHI libremente. Un software libre desempeña un papel importante en el progreso de Internet. Proporcionando la tecnología de verificación GRATIS se contribuye a los avances de IPv6.

El proyecto TAHI está integrado por dos organizaciones:

- La Universidad de Tokio
- Yokogawa Electric Corporation

TAHI proporciona:

- IPv6 ConformanceTest Tool (v6eval) [15]. Es la herramienta que implementa todas las comunicaciones para llevar a cabo los scripts
- IPv6 Conformance Test Program Package (Self Test) [23]. Son los scripts que implementan las pruebas descriptas en los documentos de Conformidad creados por el IPv6 Forum.

2.4.1 v6eval

v6eval es la plataforma para los tests de conformidad de IPv6 desarrollada por TAHI. Esto incluye no solo los archivos ejecutables, sino que además provee bibliotecas con funciones que permiten al desarrollador/tester generar sus propios tests.

2.4.2 Self Test

Self Test es la definición del conjunto de tests para Fase 2 de la certificación. Cada test se compone de un script de Perl que describe el flujo del test, y de un archivo .def que define los paquetes a enviar y/o recibir.

El resultado de cada test es un archivo HTML de log que describe todo el procedimiento del test junto a un archivo .pcap de Wireshark [17] que muestra los intercambios de paquetes entre las máquinas involucradas en el test.

2.5 Especificaciones de los tests

Existe una especificación tanto de los tests de Conformidad [21] como de los tests de Interoperabilidad [22]. Estas especificaciones han sido escritas por TAHI Project (Japón) y por la UNH (University of New Hampshire) InterOperability Lab (Estados Unidos).

Estas especificaciones, describen en detalle todos los preparativos y las interacciones que se tienen que dar entre los nodos que participan de un cierto test y para cada test se brinda:

Nombre: Se compone de una clave y un nombre de test que ayuda a identificarlo (por ejemplo: TEST V6LC.1.1.1: Version field).
Propósito: Es una pequeña frase que describe lo que el test intenta conseguir.
Referencias: Lista un conjunto de referencias a las especificaciones que permiten al tester a comprender y evaluar los resultados de un test.
Requerimientos: Especifica el software y hardware necesario para llevar a cabo el test.
Test Setup: Describe la configuración necesaria para todos los dispositivos, previo al inicio del test.
Procedimiento: Contiene las instrucciones paso a paso para llevar a cabo un test. Estos pasos incluyen cosas como habilitar interfaces, desconectar dispositivos de la red, o enviar paquetes desde uno de los nodos.
Resultados visibles: Es la lista de resultados que pueden ser observados por el tester para verificar que el nodo bajo test funciona correctamente.
Posibles problemas: Una lista de problemas conocidos que pueden ocurrir cuando se ejecuta un cierto test.

2.6 Herramientas

Las herramientas que se utilizan son los medios para la generación del ambiente de testing: VMWare, OpenSUSE, FreeBSD [4], Wireshark y Eclipse [19].

2.6.1 VMWare

Herramienta para la generación de máquinas virtuales (VM's). El uso de VMWare [16] es crucial en este proyecto, ya que de lo contrario para la conformidad necesitaríamos dos hosts físicos y en el caso de los tests de interoperabilidad necesitaríamos de cinco equipos donde 4 de ellos deben ser capaces de actuar tanto como host que como router. La virtualización en este sentido, abarata costos y simplifica el ambiente de testing (ya que se puede tener todo el ambiente creado dentro de una pc).

2.6.2 OpenSUSE

Es el sistema operativo base elegido para nuestro nodo bajo test (NUT) en la etapa de conformidad y como nodos auxiliares en la etapa de Interoperabilidad (luego veremos que a estos nodos se les llama iop1, iop2, iop3 e iop4). En este nodo va instalado OpenJDK [3] e IP4JVM y fue el sistema operativo elegido, ya que en las generaciones anteriores del proyecto funcionó con éxito. En la fase de interoperabilidad por

simplicidad se utilizan cuatro máquinas virtuales con este sistema operativo, ya que un laboratorio de certificación usaría 4 implementaciones (sistemas operativos) distintas.

2.6.3 FreeBSD

FreeBSD es el sistema operativo base para el nodo de testing (TN) para la fase de Conformidad. Las herramientas v6eval y Self Test, que permiten realizar el testing de Conformidad, fueron desarrolladas para este sistema operativo, y por lo tanto tenemos esta restricción para el TN.

2.6.4 Wireshark

Es una herramienta que permite visualizar el flujo de paquetes dentro de una cierta red. Esto es muy útil al momento de analizar errores en la fase de interoperabilidad ya que podemos ver de manera gráfica, la estructura de cada paquete que pasa por la red, y así analizar si los campos, cabezales y datos transportados, son correctos.

2.6.5 Eclipse

IDE de desarrollo en Java. Se eligió este IDE, ya que era el mismo que venía siendo utilizado por los anteriores desarrolladores de IP4JVM, y la documentación fue generada en base a él.

2.7 Ambientes de testing

2.7.1 Conformidad

En la fase de Conformidad se necesitan dos nodos. Uno que haga de nodo tester (TN) y otro que haga de nodo bajo test (NUT). Ambos nodos se crean utilizando VMWare Server 2.0.2 ya que es mucho más cómodo trabajar en una única PC que tener que manejarse con dos.

En este documento, a las dos VMs creadas para la fase de Conformidad, a una se la denomina TN y a la otra NUT. En el TN debe ser instalado el SO FreeBSD como se describe en el Anexo I. Se utiliza este sistema operativo, ya que es el único que soporta las herramientas de testing creadas por TAHI (v6eval + Self Test). Una vez instalado FreeBSD, se instalan v6eval y Self Test, como se describe en el Anexo II.

Por otra parte en el NUT, se instala OpenSUSE ya que se ha venido utilizando este SO para instalar IP4JVM y la documentación está centrada en él. Tras instalar OpenSUSE, se instala IP4JVM como se describe en el Anexo III. Una vez que sea posible levantar un Tomcat con éxito en este nodo, se estará listo para comenzar las pruebas de Conformidad.

2.7.2 Interoperabilidad

Por otra parte, la fase de testing de Interoperabilidad implica el uso de cinco nodos, con los cuales se pueden armar distintas topologías como se muestra en la Ilustración 1. A uno de ellos se le llama ip4jvm (en la etapa de Conformidad se le llamaba NUT), y a los restantes cuatro iopX (donde X = 1 al 4). Los nodos iopX, tienen instalado OpenSUSE a modo de simplificar la tarea de testing, ya que no se debe perder de vista que esto es una preparación para la certificación real, y está necesita de cuatro nodos con diferentes sistemas operativos en cada uno de ellos.

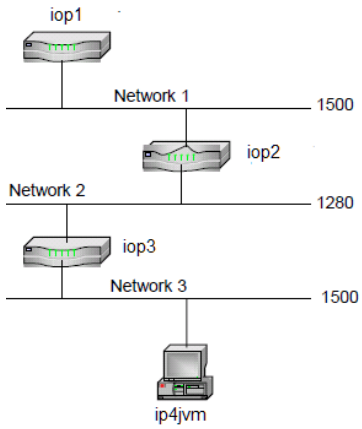

Ilustración 1 - Ejemplo de una topología

Estas cinco VMs (ip4jvm y iop1,2,3,4) están incluidas dentro de una máquina llamada iopmaster, la cual tendrá instalada la herramienta desarrollada por IRISA y continuada por Ariel Sabiguero, la cual realiza el testing de interoperabilidad.

Los tests de interoperabilidad, se llevan a cabo desde iopmaster, y se pueden observar los pormenores del proceso en el Anexo V.

3 Desarrollo del proyecto: conformidad

En esta sección detallaremos la evolución del proyecto, intentando explicar las fases en las que se dividió la etapa de conformidad, como se afrontaron dichas fases, los cambios surgidos sobre la marcha, los tiempos insumidos y los resultados obtenidos.

3.1 Planificación inicial

Al comienzo del proyecto se identificaron las distintas fases que sería necesario llevar a cabo para poder alcanzar los objetivos planteados. Dichas fases son:

Fase	**Período planeado**	**Período ejecutado**
Estudio inicial	Mayo 2010	Mayo 2010
Instalación del ambiente	Junio 2010	Junio-Julio 2010
Tests y modificaciones	Julio-Setiembre 2010	Agosto-Octubre 2010
Documentación	Octubre-Diciembre 2010	Noviembre 2010-Enero 2011

Tabla 1 - Planificación de la etapa de Conformidad

Como podemos apreciar en la Tabla 1, se produjo una pequeña desviación de lo planificado en la etapa de instalación del ambiente, que tuvo algunas complicaciones inherentes a la inexperiencia con herramientas como VMWare y un SO como FreeBSD. Las restantes fases se llevaron a cabo dentro de lo esperado.

3.1.1 Estudio inicial

La idea de esta fase fue, comprender el funcionamiento y el código desarrollado hasta el momento de IP4JVM, de las herramientas de testing de TAHI y el funcionamiento de los protocolos desarrollados. Esto incluyó las siguientes tareas:

Estudio de los manuales de usuario de v6eval [15] y Self Test [23]. Estos dos manuales, si bien eran un tanto breves, contenían claramente todos los pasos necesarios para poder utilizar la herramienta en poco tiempo.
Estudio de los documentos y del código fuente de IP4JVM. La documentación de IP4JVM [7] [8] [18] afortunadamente era muy clara y completa, sobre todo la documentación de la instalación con el "paso a paso". Al código fuente, por otra parte le faltaba un poco más de comentarios, pero los anteriores desarrolladores y el tutor se mostraron muy abiertos a responder y aclarar las dudas que iban surgiendo.
Repaso de los RFC mencionados en la sección 1.1, los cuales siguieron siendo consultados a lo largo del proyecto para solucionar ciertos errores relativos a aspectos particulares del protocolo.

Esta tarea se llevó a cabo de acuerdo a lo planificado (en el mes de Mayo 2010).

3.1.2 Instalación del ambiente

En esta etapa se replica el ambiente de testing que se describe en el documento de Conformidad nombrado en la sección 2.4. Esto incluye las siguientes tareas: creación de máquinas virtuales, instalación de v6eval e instalación de IP4JVM.

3.1.2.1 Creación de dos máquinas virtuales

Se obtiene VMWare Server 2.0.2 y el parche (sino es imposible utilizarlo en OpenSUSE), lo que permite la creación y el manejo de máquinas virtuales (VMs). Se crean dos VMs, una para el nodo bajo test (NUT) y otra para el nodo tester (TN). Esta tarea incluye el estudio del manejo de la herramienta así como las principales funciones de VMWare Server (crear una VM, configurar las interfaces de red, administración de discos virtuales y de memoria)

3.1.2.2 Instalación de v6eval y Self Test

La instalación de v6eval requiere del sistema operativo FreeBSD (se usó la versión 8.0) puesto que es el único sistema operativo en el cual se puede instalar la herramienta. Para ello se obtiene la imagen del disco de instalación. Se instala el sistema operativo en una de las VMs creadas, el software que necesita v6eval, luego se instala, se la configura y se copian los script de test disponibles en Self Test.

La instalación y forma de obtención de FreeBSD se detalla en el Anexo I. El siguiente paso es la instalación y configuración de v6eval, según se detalla en el Anexo II.
Finalmente se obtienen los script de test (llamados Self Test - última versión: 5.0), los cuales se pueden colocar en cualquier parte del TN.

3.1.2.3 Instalación de IP4JVM

Para realizar esta tarea es necesario, instalar el sistema operativo OpenSUSE 11 en la restante VM creada, junto a la más reciente versión del código fuente de OpenJDK, para finalmente instalar IP4JVM dentro de OpenJDK y compilarlos utilizando Java.

La instalación de IP4JVM se detalla en el Anexo III.

Dicha instalación, no se logró llevar a cabo de acuerdo a lo planificado debido a los siguientes problemas:

- Problemas de hardware como intentar instalar los dos sistemas operativos como particiones dentro de una misma pc (es mucho más sencillo crear 2 VMs y ponerlas en una PC).

- Problemas del kernel de FreeBSD al editar archivos de configuración. El kernel quedaba corrupto, y esto llevaba a que la VM no "booteara". Esto ocurrió varias veces lo que llevaba en última instancia a reinstalar el sistema operativo. Con la experiencia se aprendió que se podía restaurar el archivo de booteo con una copia que el SO deja disponible.
- Problemas de configuración de IP4JVM y v6eval.

Esta tarea se culminó dentro de la segunda semana de Julio.

3.1.3 Tests y modificaciones

La finalidad de esta fase es, realizar la ejecución de las pruebas, identificar los posibles errores y corregirlos. Los tests son los creados por el IPv6RL, y desarrollados por TAHI bajo la forma de Self Test, la cual utiliza el framework v6eval.

Los errores identificados y sus soluciones, se describen en el Anexo IV.

Esta fase se llevó a cabo entre la primera semana de Agosto y la segunda de Octubre (aproximadamente dos meses).

3.1.4 Documentación

El objetivo de esta fase fue, documentar todo el proceso instalación, configuración y testing; los resultados obtenidos así como las lecciones aprendidas. La documentación se fue recabando a lo largo del desarrollo del proyecto, lo que hizo más sencilla su compilación final.

La compilación de documentaciones se comenzó a mediados del mes de Octubre.

3.2 Testing

En esta sección se detalla todo el proceso de testing, la identificación de cada uno de los errores y su correspondiente solución. Este ciclo (testing=>errores=>solución) es el componente central del proyecto.

El criterio de salida de este ciclo es:
Numero de tests fallidos = 0

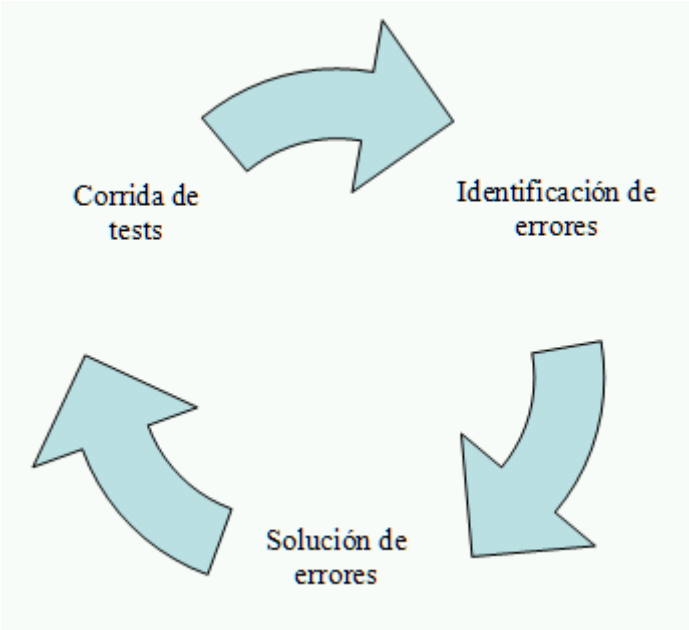

Ilustración 2 - Ciclo de testing

3.2.1 Estado inicial

Inicialmente se hizo una corrida de tests el día 8 de agosto de 2010, para verificar el estado del proyecto. Los resultados obtenidos fueron los siguientes

Sección	Correctos	Fallidos	% Éxito	Total
1- IPv6 Specification	39	15	72%	54
2- Neighbor Discovery	50	186	21%	236
3- SAA and DAD	0	45	0%	45
4- Path MTU Discovery	4	12	25%	16
5- ICMP	9	16	56%	25
Total	102	274	27%	376

Tabla 2 - Estado inicial del proyecto (Conformidad)

Esta fue una corrida inicial sin contar tests que pudieran fallar por errores de ejecución de los tests, de configuración de v6eval o de configuración de alguno de los sistemas operativos participantes. Esto claramente se debió a la inexperiencia inicial con las herramientas de testing y los archivos de configuración de dichas herramientas. Por ejemplo, una mala configuración de FreeBSD podía llevar a que paquetes que eran irrelevantes al test quedaran en la red e hiciera que algunos tests fallaran.

Otro error que en particular se dio en la sección 3, se ocasionaba por la una mala interpretación del test, ya que en estos tests primero había que inicializar los tests del lado del TN (FreeBSD) para luego inicializar la ejecución de IP4JVM y que se llevara a cabo (por ejemplo) el proceso de DAD.

Para ver más sobre este tipo de errores referirse al Anexo IV.

3.2.2 Tests y modificaciones

Los tests se dividen en cinco grupos, y cada grupo se centra en un RFC de la suite IPv6:

- IPv6 Specification
- Neighbor Discovery
- Stateless Address Autoconfiguration and Duplicate Address Detection
- Path MTU Discovery
- ICMP

A continuación se describe cada uno de estos grupos.

3.2.2.1 IPv6 Specification

Estos tests prueban todo lo competente al RFC 2460 (IPv6 Specification). Este grupo de tests está organizado en las siguientes categorías

- IPv6 Header
- IPv6 Extension Headers and Options
- Fragmentation

Se agrega el RFC 5095 (Deprecation of Type 0 Routing Headers in IPv6) que es una actualización de RFC 2460.

IPv6Header

El objetivo de estos tests es verificar que un nodo procesa y genera correctamente los campos Version, Traffic Class, Flow Label, Payload Length, Next Header y Hop Limit del cabezal IPv6.

Estos tests además verifican que un nodo transmite los mensajes ICMPv6 Parameter Problem apropiados en caso de un campo inválido o desconocido.

Los tests en la categoría IPv6 Header son los siguientes:

Group 1: IPv6 Header
Test v6LC.1.1.1: Version Field
Test v6LC.1.1.2: Traffic Class Non-Zero - End Node
Test v6LC.1.1.4: Flow Label Non-Zero
Test v6LC.1.1.5: Payload Length
Test v6LC.1.1.6: No Next Header after IPv6 Header
Test v6LC.1.1.7: Unrecognized Next Header
Test v6LC.1.1.8: Hop Limit Zero - End Node

Tabla 3 - Tests IPv6 Specification (Grupo 1)

IPv6 Extension Headers and Options

Estos tests cubren el procesamiento de opciones (options) y de los cabezales de extensión (Extension Headers), en particular Hop-by-Hop Options, Destination Options, y cabezales de ruteo (Routing headers).

Los tests de este grupo verifican que un nodo procesa y genera el campo Header Extension Length en los cabezales de extensión (Extension headers), y los campos Option Type y Option Data Length de las opciones IPv6.

Estos tests además verifican que un nodo procesa correctamente las opciones de un cabezal en orden, paquetes con un cabezal de ruteo destinado al nodo, y varios cabezales de extensión u opciones en un solo paquete.

Finalmente, los tests aseguran que un nodo genera apropiadamente mensajes ICMPv6 en respuesta a campos inválidos o desconocidos.

Group 2: Extension Headers and Options
Test v6LC.1.2.1: Next Header Zero
Test v6LC.1.2.2: No Next Header after Extension Header
Test v6LC.1.2.3: Unrecognized Next Header in Extension Header - End Node
Test v6LC.1.2.4: Extension Header Processing Order
Test v6LC.1.2.5: Option Processing Order
Test v6LC.1.2.6: Options Processing, Hop-by-Hop Options Header - End Node
Test v6LC.1.2.8: Option Processing, Destination Options Header
Test v6LC.1.2.9: Unrecognized Routing Type - End Node
Test v6LC.1.2.10: Unrecognized Routing Type - Intermediate Node

Tabla 4 - Tests IPv6 Specification (Grupo 2)

Fragmentation

Estos tests cubren los temas referidos a fragmentación en IPv6.

Verifican que un nodo realiza apropiadamente time-out al recibir paquetes fragmentados, abandona la reconstrucción de paquetes que exceden un tamaño máximo, procesa fragmentos de paquetes, y reconstruye fragmentos que se superponen. Además, se verifica que un nodo envía los mensajes ICMPv6 apropiados en caso de error.

Group 3: Fragmentation
Test v6LC.1.3.1: Fragment Reassembly
Test v6LC.1.3.2: Reassembly Time Exceeded
Test v6LC.1.3.3: Fragment Header M-Bit Set, Payload Length Invalid
Test v6LC.1.3.4: Stub Fragment Header

Tabla 5 - Tests IPv6 Specification (Grupo 3)

3.2.2.2 Neighbor Discovery

Estos tests prueban lo especificado en RFC 4861 y se organizan en las siguientes categorías

- Address Resolution and Neighbor Unreachability Detection
- Router and Prefix Discovery
- Redirect Function

Address Resolution and Neighbor Unreachability Detection

Son los tests que cubren los procesos de Address Resolution y Neighbor Unreachability Detection de IPv6.

Group 1: Address Resolution and Neighbor Unreachability Detection
Test v6LC.2.1.1: On-link Determination
Test v6LC.2.1.2: Resolution Wait Queue
Test v6LC.2.1.3: Prefix Information Option Processing, On-link Flag (Hosts Only)
Test v6LC.2.1.4: Host Prefix List (Hosts Only)
Test v6LC.2.1.5: Neighbor Solicitation Origination, Address Resolution
Test v6LC.2.1.6: Neighbor Solicitation Origination, Reachability Confirmation
Test v6LC.2.1.7: Invalid Neighbor Solicitation Handling
Test v6LC.2.1.8: Neighbor Solicitation Processing, No NCE
Test v6LC.2.1.9: Neighbor Solicitation Processing, NCE State INCOMPLETE
Test v6LC.2.1.10: Neighbor Solicitation Processing, NCE State REACHABLE
Test v6LC.2.1.11: Neighbor Solicitation Processing, NCE State STALE
Test v6LC.2.1.12: Neighbor Solicitation Processing, NCE State PROBE
Test v6LC.2.1.13: Neighbor Solicitation Processing, IsRouterFlag (Host Only)
Test v6LC.2.1.15: Invalid Neighbor Advertisement Handling
Test v6LC.2.1.16: Neighbor Advertisement Processing, No NCE
Test v6LC.2.1.17: Neighbor Advertisement Processing, NCE State INCOMPLETE
Test v6LC.2.1.18: Neighbor Advertisement Processing, NCE State REACHABLE
Test v6LC.2.1.19: Neighbor Advertisement Processing, NCE State STALE
Test v6LC.2.1.20: Neighbor Advertisement Processsing, NCE State PROBE
Test v6LC.2.1.21: Neighbor Advertisement Processing, R-bit Change (Hosts Only)

Tabla 6 - Tests Neighbor Discovery (Grupo 1)

Router and Prefix Discovery

Estos tests verifican que un host realiza Router Discovery y Prefix Discovery.

Group 2: Router and Prefix Discovery
Test v6LC.2.2.1: Router Solicitations (Hosts Only)
Test v6LC.2.2.2: Router Solicitations, Solicited Router Advertisement (Hosts Only)
Test v6LC.2.2.3: Host Ignores Router Solicitations (Hosts Only)
Test v6LC.2.2.11: Default Router Switch (Hosts Only)
Test v6LC.2.2.12: Router Advertisement Processing, Validity (Hosts Only)
Test v6LC.2.2.13: Router Advertisement Processing, Cur Hop Limit
Test v6LC.2.2.14: Router Advertisement Processing, Router Lifetime (Hosts Only)
Test v6LC.2.2.15: Router Advertisement Processing, Reachable Time
Test v6LC.2.2.16: Router Advertisement Processing, Neighbor Cache (Hosts Only)
Test v6LC.2.2.17: Router Advertisement Processing, IsRouter flag (Hosts Only)
Test v6LC.2.2.18: Next-hop Determination (Hosts Only)
Test v6LC.2.2.19: Router Advertisement Processing, On-link determination (Hosts Only)

Tabla 7 - Tests Neighbor Discovery (Grupo 2)

Redirect Function

Son los tests que verifican que un nodo procesa mensajes Redirect válidos, inválidos y sospechosos (suspicious). Además verifica que un nodo utiliza el First Hop cuando se redirige dos veces; la recepción de opciones inválidas, no tener una entrada en el Destination Cache, o cuando el primer salto (First Hop) no es alcanzable. Finalmente verifica interacciones entre las opciones del Target Link-layer Address y el Neighbor Cache.

Group 3: Redirect Function
Test v6LC.2.3.1: Redirected On-link: Valid (Hosts Only)
Test v6LC.2.3.2: Redirected On-link: Suspicious (Hosts Only)
Test v6LC.2.3.3: Redirected On-link: Invalid (Hosts Only)
Test v6LC.2.3.4: Redirected to Alternate Router: Valid (Hosts Only)
Test v6LC.2.3.5: Redirected to Alternate Router: Suspicious (Hosts only)
Test v6LC.2.3.6: Redirected to Alternate Router: Invalid (Hosts Only)
Test v6LC.2.3.7: Redirected Twice (Hosts Only)
Test v6LC.2.3.8: Invalid Option (Hosts Only)
Test v6LC.2.3.9: No Destination Cache Entry (Hosts Only)
Test v6LC.2.3.10: Neighbor Cache Updated, No Neighbor Cache Entry (Hosts Only)
Test v6LC.2.3.11: Neighbor Cache Updated from State INCOMPLETE (Hosts Only)
Test v6LC.2.3.12: Neighbor Cache Updated from State REACHABLE (Hosts Only)
Test v6LC.2.3.13: Neighbor Cache Updated from State STALE (Hosts Only)
Test v6LC.2.3.14: Neighbor Cache Updated from State PROBE (Hosts Only)
Test v6LC.2.3.15: Invalid Redirect does not Update Neighbor Cache (Hosts Only)

Tabla 8 - Tests Neighbor Discovery (Grupo 3)

3.2.2.3 Stateless Address Autoconfiguration and Duplicate Address Detection

Estos tests prueban todo lo competente al RFC 4862. Este grupo de tests está organizado en las siguientes categorías

- Address Autoconfiguration and Duplicate Address Detection
- Router Advertisement Processing and Address Lifetime

Address Autoconfiguration and Duplicate Address Detection

Son los tests que verifican que el proceso de Address Autoconfiguration y Duplicate Address Detection se ejecuta correctamente.

Group 1: Address Autoconfiguration and Duplicate Address Detection
Test v6LC.3.1.1: Address Autoconfiguration and Duplicate Address Detection
Test v6LC.3.1.2: Receiving DAD Neighbor Solicitations and Advertisements
Test v6LC.3.1.3: Validation of DAD Neighbor Solicitations
Test v6LC.3.1.4: Validation of DAD Neighbor Advertisements
Test v6LC.3.1.5: Receiving Neighbor Solicitations for Address Resolution

Tabla 9 - Tests SAA & DAD (Grupo 1)

Router Advertisement Processing and Address Lifetime

Estos tests verifican que se crean global addresses, se procesan Router Advertisements de manera correcta y se maneja correctamente la expiración de una dirección de acuerdo al IPv6 Stateless Address Autoconfiguration Specification.

Group 2: Router Advertisement Processing and Address Lifetime
Test v6LC.3.2.1: Global Address Autoconfiguration and DAD
Test v6LC.3.2.2: Address Lifetime Expiry (Hosts Only)
Test v6LC.3.2.3: Multiple Prefixes and Network Renumbering (Hosts only)
Test v6LC.3.2.4: Prefix-Information Option Processing (Hosts Only)
Test v6LC.3.2.5: Prefix-Information Option Processing, Lifetime(Host Only)

Tabla 10 - Tests SAA & DAD (Grupo 2)

3.2.2.4 Path MTU Discovery

Estos tests prueban todo lo competente al RFC 1981.

El protocolo Path MTU Discovery es una técnica para descubrir dinámicamente el PMTU de un camino en la red. La idea básica es que el nodo origen inicialmente asume que el PMTU de un camino, es el MTU del primer nodo en el camino.

Si alguno de los paquetes enviados por el camino es “muy grande” para ser reenviado, ese nodo lo descarta y devuelve un mensaje ICMPv6 Packet Too Big.

Al recibir este paquete, el nodo origen reduce el PMTU para el camino basado en el MTU del nodo que reportó el mensaje Packet Too Big.

Este proceso termina cuando el PMTU estimado es menor o igual que el PMTU almacenado.

Test v6LC.4.1.1: Confirm Ping
Test v6LC.4.1.2: Stored PMTU
Test v6LC.4.1.3: Non-zero ICMPv6 Code
Test v6LC.4.1.4: Reduce PMTU On-link
Test v6LC.4.1.5: Reduce PMTU Off-link
Test v6LC.4.1.6: Receiving MTU Below IPv6 Minimum Link MTU
Test v6LC.4.1.7: Increase Estimate
Test v6LC.4.1.8: Router Advertisement with MTU Option (Hosts Only)
Test v6LC.4.1.9: Checking For Increase in PMTU
Test v6LC.4.1.10: Multicast Destination - One Router
Test v6LC.4.1.11: Multicast Destination - Two Routers

Tabla 11 - Tests PathMTU Discovery

3.2.2.5 ICMP

Estos tests prueban todo lo descripto en RFC 4443(Internet Control Message Protocol).

Test v6LC.5.1.1: Transmitting Echo Requests
Test v6LC.5.1.2: Replying to Echo Requests
Test v6LC.5.1.3: Destination Unreachable Message Generation
Test v6LC.5.1.6: Erroneous Header Field (Parameter Problem Generation)
Test v6LC.5.1.7: Unrecognized Next Header (Parameter Problem Generation)
Test v6LC.5.1.8: Unknown Informational Message Type
Test v6LC.5.1.10: Error Condition With Multicast Destination
Test v6LC.5.1.11: Error Condition With Non-Unique Source - Unspecified
Test v6LC.5.1.12: Error Condition With Non-Unique Source - Multicast

Tabla 12 - Tests ICMP

3.2.3 Errores más reiterados

Los errores más comunes en esta etapa se pueden dividir en las siguientes categorías:

- Excepciones
- Errores de campos.
- Errores de timers
- Errores de condiciones
- Errores de configuración de v6eval
- Errores de la máquina de estados

Las excepciones se dan en respuesta a una situación inusual. Un ejemplo de excepción, se da cuando intentamos acceder a una posición inválida de un array como ocurre en el test 1.17 del Anexo IV.

Los errores de campos, se dan cuando a pesar de respetar el protocolo, se calcula mal el valor de un campo y esto provoca que una prueba falle. El test 1.8 del Anexo IV es un buen ejemplo de este tipo de errores, donde se calcula mal el campo Pointer, y esto provoca que la prueba falle.

Un ejemplo de error de timers sería el siguiente: Si un paquete debe ser retransmitido cada 3 segundos, pero se crea el código y el paquete se retransmite cada 4 segundos, este es un error . Se puede ver un ejemplo de este error en el test 2.11 del Anexo IV.

Llamamos errores de condiciones, a los errores que se ocasionan por una interpretación errónea de una parte de un protocolo, y se generar condiciones (sentencias if) incorrectas, lo que en última instancia lleva a que ciertas pruebas fallen. Se puede ver un ejemplo de este error en el Anexo IV, test 2.15.

Otra fuente de errores sobre todo en la sección 1, fue tener configuraciones erróneas de v6eval. Un ejemplo de este tipo de errores se da en el Anexo IV, Test 1.44.

Por último otro tipo de errores que se puede citar (y de pronto el más interesante), son los errores de la máquina de estados [25] del protocolo de Neighbor Discovery. En este protocolo, como se puede ver en la Ilustración 3, se manejan un conjunto de estados. Si uno de estos cambios de estado, se da de manera incorrecta, los tests fallan. Este cambio de estado incorrecto se puede dar por muchas razones: estado inicial incorrecto, una condición equivocada, un loop que no se hace, etc. En todo caso este tipo de errores es bastante complicado de debuggear porque el estado de una entrada del Neighbor Cache (lista de nodos vecinos conocidos) es una propiedad que cambia rápidamente y por ello es compleja de debuggear. La solución es verificar que dentro del algoritmo que hace el cambio de estado, se cumplan los cambios que muestra la imagen a continuación

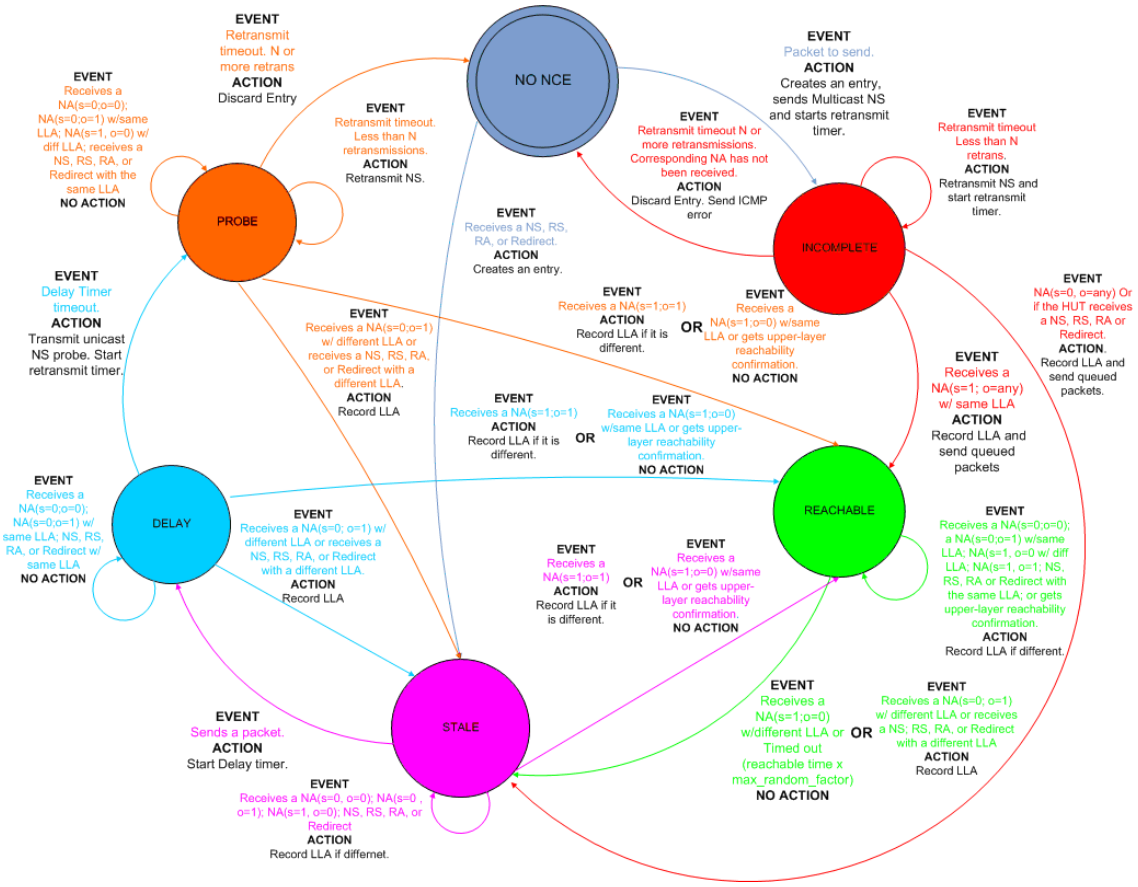

Ilustración 3 - Máquina de estados [25] del proceso de ND

3.3 Evolución de las pruebas

En la tabla 13 se muestra la evolución semanal del número de tests correctos/fallidos, y a continuación en la Ilustración 2 se muestra gráficamente dicha evolución.

SEMANA	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
PASS	102	117	153	236	322	357	363	371	376
FAIL	274	259	223	140	54	19	13	5	0
% fallidos	73%	69%	59%	37%	14%	5%	3%	1%	0%
TOTAL	376	376	376	376	376	376	376	376	376

Tabla 13 - Evolución de la etapa de Conformidad

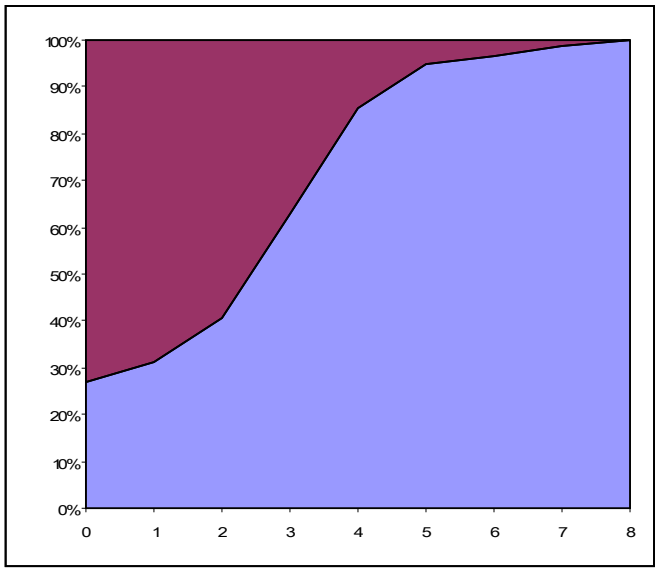

Ilustración 4 - Gráfica de la evolución de la etapa de Conformidad

En 56 días (8 semanas) se logró completar la corrección de los tests de Conformidad.

La tercera semana se podría ver como la más productiva, ya que se logró resolver 83 tests. Esto no es tan cierto ya que gran parte de esos tests resueltos correspondían a la parte de Neighbor Discovery; y varios tests se resolvían "en conjunto".

Las últimas semanas fueron las dedicadas a resolver casos de errores muy particulares, que demandaban, no solo un estudio extra de los RFC citados en esos tests, sino también cambios de código un tanto más complejos que los iniciales.

Es bastante notorio como el desarrollo inicialmente pasaba nada más que el 27% de los tests y sin embargo podía incluso completar pedidos a través de un Tomcat y manejar protocolos complejos como DHCP y NAT.

La parte que más se acoplaba a su especificación era la de las funcionalidades básicas de IPv6. Esta parte tuvo un 72% de éxito inicialmente. Las restantes secciones mostraron muchos errores, en particular la sección 2 (Neighbor Discovery) que representa más de un 50% del conjunto de tests, únicamente tuvo un 21% de éxito.

3.4 Modificaciones de código

En el Anexo IV se especifica para cada test fallido inicialmente, cuáles fueron las modificaciones necesarias para su correcto funcionamiento así como los resultados esperados, los obtenidos, el propósito del test y el procedimiento del mismo.

A modo de ejemplo, se detalla uno de los tests del anexo.

Test: Unrecognized Next Header in IPv6 Header (Multiple Values)

Propósito del test: Este test verifica que un nodo utilizando IP4JVM, genera una respuesta apropiada en caso de recibir un paquete con el campo Next Header con un valor desconocido.

Resultados esperados:
El NUT debe enviar un ICMPv6 Parameter Problem message al TN tal que:
ICMPv6 Code = 1 (unrecognized Next Header type encountered).
ICMPv6 Pointer field = 0x06 (offset of the Next Header field).

Resultado obtenido:
Se recibió el siguiente paquete:

```
| Packet_IPv6                        (length:94)
| | Hdr_IPv6                           (length:40)
| | | Version                             = 6
| | | TrafficClass                        = 0
| | | FlowLabel                           = 0
| | | PayloadLength                       = 54
| | | NextHeader                          = 58
| | | HopLimit                            = 255
| | | SourceAddress                       = fe80::20c:29ff:fe73:d259
| | | DestinationAddress                  = fe80::20c:29ff:feac:521a
| | ICMPv6_ParameterProblem            (length:54)
| | | Type                                = 4
| | | Code                                = 1
| | | Checksum                            = 7733 calc(7733)
| | | Pointer                             = 4294967288
| | | Payload                           (length:46)
| | | | data                              =
| | | |   60000000 00068fff  fe800000 00000000  020c29ff feac521a
| | | |   fe800000 00000000  020c29ff fe73d259  00000000 0000
```

Ilustración 5 - Paquete recibido en un test de Conformidad

En este caso el test falla ya que el valor del campo Pointer (resaltado) dentro del paquete ICMPv6 es 0xFFFFFFF8 (-8 en notación decimal) que es distinto del esperado (0x06).

Solución: Modificar la clase ip4jvm.net.protocols.IPv6Protocol (línea 162)

Para que el test pase se debe modificar la línea 162 que es la que asigna el valor de nextHeadOffset, de forma que el valor del campo Pointer (dado por la expresión nextHeadOffset – netMsg.getOffset()) sea el correcto.

Código original :

```
159             if (header.getExtraHeadersCount() != 0) {
160                 nextHeadOffset = header.getLastExtraHeaderOffset();
161             } else {
162                 nextHeadOffset = IPv6ProtocolHeader.VER_TRAFF_FL_WIDTH + IPv6ProtocolHeader.LENGTH_WIDTH;
163             }
164             sendParameterProblem(ns.getNetconfig(), ICMPv6PProblem.UNRECOGNIZE_NEXTHEADER,
165                                  netMsg.getBytesfromOffset(), nextHeadOffset - netMsg.getOffset());
```

Ilustración 6 - Fragmento de código original de la clase ip4jvm.net.protocols.IPv6Protocol

Código correcto:

```
159             if (header.getExtraHeadersCount() != 0) {
160                 nextHeadOffset = header.getLastExtraHeaderOffset();
161             } else {
162                 nextHeadOffset = IPv6ProtocolHeader.VER_TRAFF_FL_WIDTH + IPv6ProtocolHeader.LENGTH_WIDTH + netMsg.getOffset();
163             }
164             sendParameterProblem(ns.getNetconfig(), ICMPv6PProblem.UNRECOGNIZE_NEXTHEADER,
165                                  netMsg.getBytesfromOffset(), nextHeadOffset - netMsg.getOffset());
```

Ilustración 7 - Fragmento modificado de código de la clase ip4jvm.net.protocols.IPv6Protocol

Con esta modificación, el valor del campo Pointer se calcula correctamente, y así el test es exitoso, mientras que los restantes tests mantienen sus resultados iniciales.

Clases modificadas: ip4jvm.net.protocols.IPv6Protocol

4 Desarrollo del proyecto: interoperabilidad

En esta sección detallaremos la evolución de la fase de interoperabilidad, intentando detallar las fases en las que se dividió el mismo, como se afrontaron dichas fases, los cambios surgidos sobre la marcha, los tiempos insumidos y los resultados obtenidos.

4.1 Planificacion inicial

Al comienzo del proyecto se identificaron las distintas fases que sería necesario llevar a cabo para poder alcanzar los objetivos planteados. Dichas fases son:

Fase	Período planeado	Período ejecutado
Estudio inicial	Mayo 2010	Marzo 2011
Instalación del ambiente	Junio 2010	Abril-Mayo 2011
Conformidad (parte 2)		Junio 2011
Tests y modificaciones	Julio-Setiembre 2010	Julio-Octubre 2011
Documentación	Octubre-Diciembre 2010	Noviembre 2011

Tabla 14 - Planificación de la etapa de Interoperabilidad

Como se puede apreciar en la Tabla 14, surgió una etapa que no estaba considerada en la planificación. Esto se debió a que otro grupo de estudiantes realizó en paralelo modificaciones en el código de IP4JVM, lo que introdujo errores a los tests de Conformidad y llevó a que nuevamente se tuviera que lograr un 100% de éxito en la etapa de Conformidad (de ahí el nombre de la etapa – Conformidad parte 2–).

La diferencia sustancial entre lo planeado y lo ejecutado se dio porque las etapas de Conformidad y de Interoperabilidad no pudieron ser llevadas a cabo en paralelo como se pretendía inicialmente.

4.1.1 Estudio inicial

La idea de esta fase fue, comprender el proceso de testing de interoprabilidad y evaluar las alternativas para llevarlo a cabo. Se plantearon dos opciones para la realización de este proceso: Ejecutarlo manualmente ó con una herramienta semi-automática desarrollada por terceros

Esta tarea se llevó a cabo de acuerdo a lo planificado.

4.1.2 Instalación del ambiente

En esta etapa hay que replicar el ambiente de testing que se describe en el documento de Interoperabilidad nombrado en la sección 2.4. Esto incluyó las siguientes tareas:

Creación de una máquina con 4 VMs (Virtual Machines - VMs)

Debido a que la fase de interoperabilidad implica el uso de 5 máquinas interconectadas, resulta mucho más sencillo manejarse en un ambiente virtualizado.
Para esto se crearon 4 máquinas que permiten crear las topologías de red especificadas en los tests junto además de la máquina anteriormente creada que tiene instalado IP4JVM (la máquina bajo test).

Migración de estas VMs a una nueva máquina física.

Ya que esta máquina fue creada por el tutor del proyecto y esta estaba en su domicilio se hacía muy complicado realizar las corridas de test por no tener acceso a la máquina física. Para solucionar esto se migraron las VM a una nueva máquina.

Esta tarea culminó dentro del mes de Junio.

4.1.3 Conformidad (Parte II)

Debido a que en paralelo, otro grupo realizó modificaciones al código de IP4JVM para introducir nuevas características de MIPv6, se debió correr los tests de conformidad nuevamente y resolver los problemas asociados al nuevo código.

Este nuevo código introdujo algunos errores en la etapa de conformidad, pero con un buen trabajo en conjunto no hubo mayores complicaciones para culminar con esta etapa con éxito rápidamente.

Esto insumió aproximadamente el mes de Mayo de 2011 y los errores surgidos así como las modificaciones necesarias se desarrollan en el Anexo IV, Capítulo 2.

4.1.4 Tests y modificaciones

La idea de esta fase fue, realizar las corridas de tests, identificar los posibles errores y corregirlos. Los tests son los creados por el IPv6RL.

Esta fase se llevó a cabo entre la primera semana de Junio y la segunda de Octubre. Esto se debió a que algunos errores fueron muy complicados de debuggear, hallar y resolver.

Estos errores y sus soluciones se detallan en el Anexo V.

4.1.5 Documentación y correcciones a la antigua documentación

La idea de esta fase fue, documentar todo el proceso: instalación, configuración y testing de interoperabilidad; los resultados obtenidos, las lecciones aprendidas. Además se modificaron las documentaciones anteriores a modo de actualizarlas.

La compilación de documentaciones se comenzó a mediados del mes de Octubre.

4.2 Testing

En esta sección se detalla todo el proceso de testing de interoperabilidad, la identificación de errores y su correspondiente solución. El proceso es similar al expuesto en el capítulo 3.2, con el agregado que si realizamos modificaciones al código además de probar los tests de interoperabilidad, debemos probar los tests de conformidad, para descartar la inyección de nuevos errores.

4.2.1 Estado inicial

Inicialmente se hizo una corrida de tests, para verificar el estado del proyecto. Los resultados obtenidos fueron los siguientes

1	1.1.A	OK	11	1.1.E	FALLA	21	1.3.A	OK	31	1.6.A	OK
2	1.1.B	OK	12	1.1.F	FALLA	22	1.3.B	OK	32	1.6.A	OK
3	1.1.C	FALLA	13	1.2.A	OK	23	1.3.C	OK	33	1.6.C	FALLA
4	1.1.A	OK	14	1.2.B	OK	24	1.3.A	OK	34	1.6.C	FALLA
5	1.1.B	OK	15	1.2.A	OK	25	1.3.B	OK	35	Setup	OK
6	1.1.C	FALLA	16	1.2.B	OK	26	1.3.C	OK	36	1.6.D	FALLA
7	1.1.D	FALLA	17	1.2.C	OK	27	1.4	OK	37	1.6.D	FALLA
8	1.1.E	FALLA	18	1.2.D	OK	28	1.4	OK			
9	1.1.F	FALLA	19	1.2.C	OK	29	1.5	OK			
10	1.1.D	FALLA	20	1.2.D	OK	30	1.5	OK			

Tabla 15 - Estado inicial de la etapa de Interoperabilidad

En la tabla 16 podemos observar la evolución de los tests aprobados (PASS), los fallidos (FAIL), así como el porcentaje de fallidos, a lo largo de las 17 semanas que insumió el testing de interoperabilidad.

SEMANA	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
PASS	26	28	28	28	28	28	28	34	34
FAIL	10	8	8	8	8	8	8	2	2
% fallidos	28%	22%	22%	22%	22%	22%	22%	6%	6%
TOTAL	36	36	36	36	36	36	36	36	36

SEMANA	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
PASS	34	34	34	34	34	34	34	34	36
FAIL	2	2	2	2	2	2	2	2	0
% fallidos	6%	6%	6%	6%	6%	6%	6%	6%	0%
TOTAL	36	36	36	36	36	36	36	36	36

Tabla 16 – Evolución de la etapa de Interoperabilidad

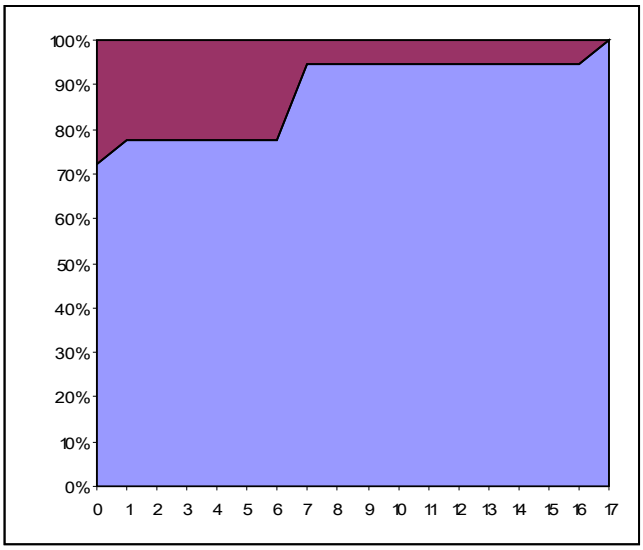

Ilustración 8 - Gráfica de la evolución de la etapa de Interoperabilidad

Algo positivo que se puede rescatar de esta gráfica es que se comenzó las correcciones con un 72% de tests correctos. Probablemente la corrección de errores debido a las pruebas de conformidad aportó calidad a la implementación, pero no cubrió en su totalidad, las pruebas que se realizan en interoperabilidad.

Por otra parte, se puede observar como entre la semana 1 y 6, y luego entre la 7 y 16 se ven dos mesetas importantes. Esto se debió a que dos errores específicos, llevaron más de lo esperado, sobre todo los últimos dos errores (tests 8 y 11) llevaron unas 11 semanas de resolución, ya que el error fue muy complejo de resolver.

4.2.2 Errores y modificaciones

En el Anexo V se especifica para cada test fallido inicialmente, cuáles fueron las modificaciones necesarias para su correcto funcionamiento así como los resultados esperados, los obtenidos, el propósito del test y el procedimiento del mismo..

A modo de ejemplo, se detalla uno de los tests del anexo.

Test 1.1.E: ICMPv6 Echo Interoperability - Global Unicast Address (Host vs Router)

Proposito del test: Verificar que se logra intercambios de Echo Request-Echo Reply en ambas direcciones.

Resultados esperados:

18. Enviar ICMPv6 Echo Requests del TAR-Host1 al Global unicast address de TARRouter1.
19. Observar los paquetes en Network1.
20. Enviar ICMPv6 Echo Requests de TAR-Router1 al Global unicast address de TAR-Host1.

21. Observar los paquetes en Network1.

Resultados esperados:

19: El TAR-Router1 debe recibir todos los ICMPv6 Echo Requests enviados desde TARHost1 y responder con ICMPv6 Echo Replies. El Source Address del Echo Reply debe ser igual al Destination Address del Echo Request, y el Destination Address del Echo Reply debe ser igual al Source Address del Echo Request.
Step 21: TAR-Host1 debe recibir todos los ICMPv6 Echo Requests enviados desde TARRouter1 y responder con ICMPv6 Echo Replies. El Source Address del Echo Reply debe ser igual al Destination Address del Echo Request, y el Destination Address del Echo Reply debe ser igual al Source Address del Echo Request.

Resultado obtenido:

Al intentar hacer ping hacia/desde ip4jvm se produce el siguiente error:
Connect: Network is unreachable

Solución:

La solución a este error, aunque sencilla en términos de la solución en sí, fue muy complicada de encontrar. Primero se intento atribuirla a errores en ip4jvm. Se debuggeo el código para ver si había algún problema para este test en particular, con resultados negativos.

Luego, se descubrió que si se ejecuta este test únicamente, se corre correctamente. Sin embargo, si se corren otros tests antes, no se corre correctamente. Esto sugería que el error debía estar del lado de la máquina tester iopmaster y no de la máquina ip4jvm.

Se intentó por varios caminos identificar el error hasta que se llegó a la conclusión de que el error radicaba en que no se daba correctamente de baja la interfaz iop1.

```
ifconfig eth0 down
ifconfig eth0 0.0.0.0
```

Resultaba que también se necesitaba dar de baja la dirección global creada en uno de los primeros pasos

Para esto se agregó en los scripts de los tests que utilizan direcciones globales dentro de ip4jvm/interop/out/iopmaster/, la siguiente línea de código:

```
ifconfig eth0 del direccion_global
```

5 Conclusiones y trabajo futuro.

La conclusión primaria es que se consiguió el objetivo impuesto que consistía en prepararse para certificar IP4JVM respecto a los tests de conformidad e interoperabilidad desarrollados por la IPv6RL.

Las correcciones en la etapa de Conformidad, en sí, no fueron muy complejas (salvo un par de casos puntuales) y constituían errores de interpretación del protocolo u errores en la lógica de programación. Sobre el final, si, se pudieron ver algunos errores más complejos de resolver y que implicaban la modificación de varias clases para su correcto funcionamiento.

La interoperabilidad, por otra parte, resultó más compleja y extensa de lo esperado. Primero se sumaron algunos problemas con el ambiente de testing (se comenzó a hacer el testing usando una máquina pero luego se pasó a otra máquinas por lo explicado en 4.1.2), al tiempo que otro grupo de estudiantes comenzó a realizar modificaciones en el código agregando MIPv6 (por lo que hubo que realizar los tests de conformidad nuevamente). Luego los errores de interoperabilidad no resultaron tan sencillos de resolver como los de conformidad, ya que no correspondían solamente a problemas en la máquina bajo test, sino que resultó que la máquina que realizaba los tests también tenía sus propios problemas. Además de lo mencionado surgieron algunos problemas personales que al ser un proyecto de índole individual, obviamente impactaron en el tiempo de realización de esta etapa.

En cuanto a las enseñanzas de este proyecto, lo que más se rescata es trabajar en algo novedoso y de actualidad como lo es IPv6. Un protocolo que en el mediano plazo va a ser muy utilizado en informática, debido al ya sabido problema que surgió con IPv4.

Aprender los detalles de los protocolos, es otra cosa sin dudas interesante para aprender cómo se lleva a cabo todo el proceso de Neighbor Discovery, DAD, los intercambios de mensajes, etc.

Configurar un ambiente de testing en un sistema operativo como FreeBSD fue sin dudas todo un reto, sobre todo cuando FreeBSD caía en errores fatales y había que reconfigurar el kernel para seguir trabajando (hubieron algunas re-instalaciones desde 0).

En cuanto al trabajo futuro, y siguiendo por la rama del testing; sin dudas sería interesante realizar todo el conjunto de tests de conformidad para los restantes protocolos, como ser DHCPv6, MIPv6, e IPSec (previa implementación completa).

También podría resultar interesante realizar una herramienta que implemente los tests de interoperabilidad de manera automática similar a lo que hace v6eval para los tests de conformidad. Esto se podría lograr modificando la herramienta ya existente desarrollada por IRISA y mejorada por el tutor de este proyecto, Ariel Sabiguero.

6 Difusión del proyecto

En marzo de 2011, se presentó el proyecto en el SASE (Simposio Argentino de Sistemas Embebidos) que es un evento anual que reúne a la comunidad académica y a la industria en torno a la temática de los sistemas embebidos. Dicho evento se llevó a cabo en la Facultad Regional Buenos Aires de la Universidad Tecnológica Nacional (UTN). El proyecto contó con una mención de honor por parte de un tribunal compuesto de cuatro ingenieros.

http://www.sase.com.ar/2011/concursos/ganadores-del-concurso-de-proyectos-estudiantiles/

En el mes de Agosto, el proyecto estuvo presente en el Tech Day, un evento organizado por Zonamerica, que expuso lo más importante en tecnología y contó con la presencia de aproximadamente 2.000 personas, muchas de las cuales se detuvieron a observar el proyecto. Además, dos medios de prensa muy importantes como El País y El Observador, realizaron una pequeña entrevista.

Por último en Noviembre, se presentó el proyecto en el evento “Ingeniería de Muestra 3” que organizó la Facultad de Ingeniería junto a otros tantos proyectos de fin de carrera.

7 Glosario

DAD (Duplicate Address Detection): mecanismo por el cual un nodo que está obteniendo su dirección de red, puede detectar si esa dirección es única o no.

Framework: estructura de soporte que abstrae de complejidades subyacentes y se utiliza para implementar funcionalidades de una manera estructurada y más simple.

Host: nodo de red con nombre identificador. O máquina huésped de determinado software.

HTTP (Hyper-Text Transfer Protocol): protocolo utilizado por Internet para la transferencia de páginas web.

ICMP (Internet Control Message Protocol): sub protocolo de IP utilizado para el envío y gestión de mensajes de control.

IPv6: Internet Protocol version 6

IPv6RL (IPv6 Ready Logo): organización que define los tests de conformidad de IPv6.

IRISA: Institute de Recherche en Informatique et Systèmes Aléatoires

MTU (Maximum Transmission Unit): tamaño en bytes de la unidad de datos más grande que puede enviarse usando un Protocolo de Internet. En IPv6 el mínimo MTU es 1280 bytes.

NUT (Node Under Test): Nodo bajo prueba. En el contexto de este proyecto, un nodo corriendo IP4JVM.

Perl: lenguaje de programación, en el cual están implementados los tests de v6eval.

PMTU (Path MTU): MTU de un camino dentro de una red.

RFC (Request For Comments): documento cuyo contenido es una propuesta oficial para un nuevo protocolo de red, que se explica con todo detalle para que en caso de ser aceptado pueda ser implementado sin ambigüedades.

SAA (Stateless Address Autoconfiguration): mecanismo por el cual un nodo puede generar su propia dirección de red.

TAHI: organización que implementa herramientas y frameworks de testing IPv6 (por ejemplo v6eval).

TN (Tester Node): Nodo de la topología que comienza el test. En el contexto de este proyecto, el nodo corriendo v6eval.

v6eval: framework para testing de IPv6.

VM (Virtual Machine): máquina virtual refiere a la utilización de un software que simula la existencia de una máquina real en un host.

Wireshark: herramienta que permite visualizar el tráfico de una red.

8 Referencias

[1] IPv6 Ready Logo - http://www.ipv6ready.org/

[2] TAHI Project, tests de conformidad e interoperabilidad para IPv6 - http://www.tahi.org/

[3] Open JDK - http://openjdk.java.net/

[4] Free BSD - http://www.freebsd.org/

[5] FreeBSD Developers Handbook - http://www.freebsd.org/doc/en/books/developers-handbook/ipv6.html

[6] The Internet Engineering Task Force (IETF) - http://www.ietf.org/rfc.html

[7] Documentación IP4JVM - http://www.fing.edu.uy/~asabigue/prgrado/2007IP4JVM_Abelenda_Corrales.pdf
Roger Abelenda, Ignacio Corrales.

[8] Documentación IP4JVM - http://www.fing.edu.uy/~asabigue/prgrado/scasso_techera/IP4JVM.pdf
Leandro Scasso, Marcos Techera.

[9] Open Suse – http://opensuse.org/

[10] RFC 1981 - Path MTU Discovery for IP version 6 -http://www.ietf.org/rfc/rfc1981.txt (Agosto 1996 - J. McCann, S. Deering, J. Mogul)

[11] RFC 2460 - Internet protocol, Version 6 (IPv6) Specification - http://www.ietf.org/rfc/rfc2460.txt (Diciembre 1998 - S. Deering, R. Hinden)

[12] RFC 4443 - Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification - http://www.ietf.org/rfc/rfc4443.txt
(Marzo 2006 - A. Canta, S. Deering, M. Gupta)

[13] RFC 4861 - Neighbor Discovery for IP version 6 (IPv6) - http://www.ietf.org/rfc/rfc4861.txt (Septiembre 2007 - T. Narten, E. Nordmark, W. Simpson, H. Soliman)

[14] RFC 4862 - IPv6 Stateless Address Auto configuration http://www.ietf.org/rfc/rfc4862.txt (Septiembre 2007 - S. Thomson, T. Narten, T. Jinmei)

[15] IPv6 Verification Tool Users Manual – http://www.tahi.org/conformance/doc/tool-2.3/v6eval-e.pdf

[16] VMware, sistema de virtualización por software - http://www.vmware.com/

[17] Wireshark, analizador de tráfico de redes - http://www.wireshark.org/

[18] Documentación IP4JVM - https://gforge.inria.fr/plugins/scmsvn/viewcvs.php/doc/PrimeraIteracion/?root=ip4jvm - Laura Rodríguez

[19] Eclipse - an open development platform - http://www.eclipse.org/

[20] Computer Networks – Third Edition - Andrew S. Tanenbaum – Pearson Education – 1996 - ISBN: 0133942481

[21] Especificaciones de los tests de Conformidad - Ipv6Forum, TAHI Project (Japan) & UNH InterOperability Lab (USA) - http://www.ipv6ready.org/docs/Core_Conformance_Latest.pdf

[22] Especificaciones de los tests de Interoperabilidad – Ipv6Forum, TAHI Project (Japan) & UNH InterOperability Lab (USA) - http://www.ipv6ready.org/docs/Core_Interoperability_Latest.pdf

[23] Self Test – Conjunto de scripts para Testing de Conformidad – http://www.tahi.org/logo/release/Self_Test_5-0-0.tgz

[24] RFC 5095: Deprecation of Type 0 Routing Headers in IPv6 (Updates 2460) - http://www.ietf.org/rfc/rfc5095.txt (Diciembre 2007 - J. Abley, Afilias, P. Savola)

[25] Neighbor Discovery State Machine for the Reachability State – A. Gadzik, T. Peterson – http://www.iol.unh.edu/services/testing/ipv6/training/ND_statemachine.gif

[26] Sitio personal de Ariel Sabiguero - http://www.fing.edu.uy/~asabigue/