Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Proyecto de Grado

Modelado e implementación de una suite de tests de interoperabilidad en IPv6

Leonardo M. Castillo Cirullo

2010
Montevideo – Uruguay

Tribunal:

Guzmán Llambías
Carlos Luna
Carlos Martínez

Tutor:

Ariel Sabiguero Yawelak

Resumen

El IPv6 Ready Logo (v6RL) elabora especificaciones de suites de casos de prueba, para poder determinar si un dispositivo esta preparado para ser utilizado en una red IPv6. Las especificaciones se publican como documentos en idioma Inglés que describen las pruebas a realizar en lenguaje natural.

Este trabajo aborda el problema de la formalización de la especificación de los casos de prueba, proponiendo un conjunto de abstracciones que abordan el modelado de la suite del v6RL.

En base a una heramienta existente, se implementó un intérprete de dicho modelo que es capaz de ejecutar una suite, a partir del modelo contra una implementación. Se modeló la especificación de las fases 1 y 2, versión 4.0.1, de los tests de interoperabilidad.

La implementación elaborada en este trabajo, puede ser desplegada en una única máquina, utilizando tecnologías de virtualización. De esta manera, se mejoran costos, aumenta la practicidad, se disminuyen las tareas manuales, bajando la probabilidad de errores y reducción de recursos de hardware requeridos.

Palabras clave: testing de interoperabilidad, modelado de interoperabilidad, IPv6, IPv6 Ready Logo, PERL.

Índice de contenido

Índice de ilustraciones

1. Introducción.

El presente documento es una propuesta de modelado e implementación para el testing de Interoperabilidad en IPv6. El mismo es elaborado en calidad de proyecto de grado de la carrera de Ingeniería en Computación de la Facultad de Ingeniería (http://www.fing.edu.uy/inco) de la Universidad de la República Oriental del Uruguay (UdelaR).

Nuestra versión (aún actual) para la construcción en Internet, el IPv4, fue estandarizado en 1981 (RFC 791 [12]). Muchas de las especificaciones iniciales se volvieron obsoletas por el crecimiento de Internet, empezándose a hacer notorias dichas circunstancias en la década del 90.

Así es, como en 1995 se publica la especificación de lo que podríamos llamar la siguiente generación de protocolos de Internet, IPv6 (RFC 1883 [13]).

Que es cosa de tiempo la necesidad de migrar nuestro “actual” Internet, es decir IPv4, al “nuevo” Internet, IPv6, es algo que aún podemos seguir afirmando, quizás sin el componente de dramatismo que en algún momento se le imprimió, pero es algo perfectamente lógico suponer que más temprano que tarde, IPv6 terminará por sustituir nuestro actual IPv4.

Es decir, tarde o temprano nos toparemos con la necesidad de IPv6, y no será únicamente una cuestión de gusto. Cuando eso ocurra, es que necesitaremos nuestro software y nuestro hardware listo para ello.

Es por esa razón que deberemos confiar en que nuestros proveedores satisfagan los requerimientos solicitados. Los tests de conformidad e interoperabilidad para IPv6, son una suite de tests cuya finalidad es asegurar que el dispositivo a testear esté listo para interactuar con dispositivos de otros proveedores, es decir, que cuando llegue IPv6, nosotros estaremos preparados.

El IPv6 Ready Logo Program es un conjunto de pruebas de conformidad e interoperabilidad cuyo propósito es generar confianza, demostrando que IPv6 está disponible y listo para ser utilizado.

Las herramientas de modelado nos permiten representar un sistema existente, e incluso uno a ser creado, con la facilidad que los costos de cualquier modificación, en líneas generales, son menores que los producidos por modificar el sistema en sí mismo.

Los tests de interoperabilidad implican la interacción entre distintos componentes, en este caso de hardware, que nos aseguren la posibilidad de establecer una comunicación entre ambos.

Dentro de este documento, nos fijamos como meta aunar ambos requerimientos, el modelado y la interoperabilidad, realizando esta tarea para la suite IPv6 de IPv6 Ready Logo.

Es así que, procedemos a fijar como meta de este documento, la obtención, no sólo de un modelado, sino además de una potencial implementación que satisfaga la suite de tests elaborados por IPv6 Ready Logo.

Metodología seguida para la realización del proyecto.

Los pasos llevados a cabo durante la realización de este proyecto pueden determinarse, en un primer acercamiento, en base al contenido de los capítulos de éste documento. Se comienza por investigar la existencia de herramientas de modelado que puedan ser aplicables para nuestras necesidades. Posteriormente, se procede a la formulación de una propuesta de modelado capaz de satisfacer los requerimientos necesarios. Con estas tareas completadas, continuamos con la implementación de una herramienta capaz de satisfacer el modelado propuesto.

Público objetivo y conocimientos previos.

El contenido de este documento principalmente esta dirigido a estudiantes y profesionales del área de Computación, Electrónica y Comunicación de Datos, no obstante, la solución e implementación elaborada, fue realizada con la intencionalidad de brindar la suficiente documentación como para que este documento este al alcance de cualquier interesado con conocimientos en el área de la programación.

Pese a no ser requisito excluyente, es conveniente, para facilitar la lectura y comprensión del documento, conocimientos generales de redes, comunicación de datos (IPv6), programación y modelado.

Organización del documento.

El contenido de este documento, detalla los pasos llevados a cabo para la posible utilización de un modelado, la implementación obtenida, y las tareas que quedan pendientes a futuro, para de esta manera dotar la implementación realizada de un mayor valor.

Para la facilidad en la comprensión y seguimiento de este documento, se realizan las siguientes consideraciones: en los dos capítulos que a continuación se detallan, se trata de reseñar la búsqueda de un modelado ya existente. En el siguiente capítulo el modelado utilizado. A continuación, la implementación y pruebas realizadas, y en los últimos capítulos, las conclusiones y trabajos a futuro.

Por tanto, en el caso que el interés en la lectura de este documento sea en lo referente al modelado, bastaría con leer el capítulo “Propuesta de modelado”.

Si en cambio, el interés es enfocarse en la implementación llevada a cabo, se recomienda realizar al menos una lectura del capítulo “Propuesta de modelado”, y enfocarse en el contenido del capítulo “Implementación”. Claro está, se sugiere asimismo la lectura del Apéndice B.

Si se desea tomar este documento como punto de partida para extender alguna funcionalidad, se debería seguir la recomendación previa, y agregar los capítulos finales del documento.

2. Estado del arte

En esta sección veremos un poco de historia de IPv6, así como, la relación entre IPv6 Ready Logo y las suite de tests de conformidad e interoperabilidad.

2.1 Acerca del protocolo IPv6 y sus pruebas.

La IETF (Internet Engineering Task Force, http://www.ietf.org/) es una organización sin fines de lucro que trabaja en la estandarización de los protocolos de IPv6.

Estos estándares son desarrollados por grupos de trabajo de IETF, principalmente por el IPv6WG (IPv6 Working Group), el cual es el encargado de definir la especificación y la estandarización de la versión 6 del protocolo de Internet.

Hay muchas organizaciones dedicadas a la promoción de IPv6. El principal consorcio a nivel mundial es el IPv6 Forum (http://www.ipv6forum.com/), organización formada por proveedores líderes en Internet, investigadores y educadores en redes (NREN).

Muchas organizaciones regionales son promotoras activas de IPv6, como la "IPv6 Task Forces" (IPv6TF), en Norteamérica la Nav6TF, en Europa la European IPv6TF y el concilio de promoción de IPv6 en Asia (Japanese IPv6 Promotion Council).

El proyecto de trabajo más importante para el desarrollo de IPv6 es "Widely Integrated Distributed Environment (WIDE), un consorcio de investigación formado por la industria, instituciones públicas y academias de Japón fundado en 1985. Su enfoque de operaciones es Internet y es uno de los vanguardistas en el desarrollo de proyectos en IPv6, como KAME http://www.kame.net/, (el cuál provee en forma gratuita un IPv6 operativo e IPv6 Stack para las variantes *BSD de Unix, proveyendo código incluído en el Kernel), UniverSAl playGround for IPv6, http://www.linux-ipv6.org/ (USAGI, el cuál provee una implementación de IPv6 e IPSec para Linux) y TAHI, http://www.tahi.org/ (desarrolla y provee tecnología para IPv6, trabaja en forma colaborativa con las dos anteriores y proporciona test de conformidad e interoperabilidad para IPv6, y es una de las bases del IPv6 Ready Logo Program). Como se puede apreciar, estos proyectos trabajan en conjunto para ofrecer gratuitamente y en modalidad de open-source código y herramientas IPv6 para diferentes plataformas.

¿Qué es IPv6 Ready Logo Program?

Según su propia definición: "El IPv6 Forum es un consorcio a nivel mundial, cuyo enfoque clave es proporcionar orientación técnica para el desarrollo de IPv6, puesto en marcha mediante el IPv6 Ready Logo Program (testing de conformidad e interoperabilidad). IPv6 Ready Logo Program es un conjunto de tests de conformidad e interoperabilidad destinados a incrementar la confianza del usuario, demostrando que IPv6 ya está disponible y pronto para ser utilizado." (http://www.ipv6ready.org/?page=about)

Surge como forma de acelerar el desarrollo y homologación de dispositivos IPv6. Su premisa se basa en la idea de la existencia de un logo el cual indica que un dispositivo es apto, y además esta disponible, para trabajar sobre IPv6.

Para obtener la certificación de IPv6 Ready Logo, los dispositivos deben demostrar conformidad con la especificación, así como probar interoperabilidad con otras implementaciones. Es decir, deben ser sometidos a un conjunto de pruebas o tests, para asegurar conformidad e interoperabilidad. Los posibles dispositivos a ser certificados son: host, routers o dispositivos especiales.

Test de Conformidad e Interoperabilidad.

Todo producto fabricado y que desee contar con el logo de IPv6 Ready, debe satisfacer ambos tests.

El test de conformidad asegura que un producto cumple con las especificaciones en las cuáles se basó (es decir el dispositivo es conforme a las especificaciones).

El test de interoperabilidad busca comprobar que el mismo es capaz de "interactuar" con productos de fabricantes y/o modelos distintos, es decir, no solo cumple con las especificaciones, sino que además es capaz de comunicarse con otros.

En líneas generales no deberíamos considerar como una posibilidad que un dispositivo aprobará un test de conformidad y no el test de interoperabilidad, pero en la práctica y dado que los tests básicamente son un conjunto de pruebas a realizar, podríamos encontrar casos en los que nos enfrentáramos a situaciones de este tipo, a modo de ejemplo dispositivos de un fabricante conformes a la especificación que fueran capaces de comunicarse entre sí, e incapaces de comunicarse con dispositivos de otro/s fabricantes.

Fases de IPv6 Ready.

Existen tres fases, asociadas a tres logos del IPv6 Ready que determinan el grado de aceptación del producto con respecto al grado de conformidad determinado por el IPv6 Logo Committee (v6LC).

Fase 1 (Silver Logo) 

. Fase iniciada en setiembre del 2003, la cumplen los dispositivos que soportan funcionalidades básicas de IPv6 con mínimas garantías. El dispositivo debe pasar el testeo del 100% de los protocolos obligatorios de IPv6 y debe también, poder interoperar con al menos cuatro dispositivos de otros fabricantes, incluyendo dos host y dos routers. Los RFC comprendidos en esta fase de validación son: RFC 2460 (especificación base del protocolo IPv6), RFC 4291 (arquitectura de direccionamiento), RFC 4861 (especificación de Neighbor Discovery, es decir descubrimiento de vecinos), RFC 4862 (especificación de auto-configuración de direcciones), RFC 4443 (ICMP para IPv6).

Fase 2 (Gold Logo) 

. Iniciada en febrero del 2005, se solicitan los mismos requerimientos que para la fase 1, además de un conjunto más preciso de tests de conformidad e interoperabilidad, que permitan validar cosas tales como, PMTUD (path

maximun tranmission unit discovery), IPSec o MIPv6. Los RFC comprendidos en esta fase de validación son: los de la fase 1 más RFC 1981 (Path MTU Discovery para IPv6).

Para esta fase existen, además, cuatro logos opcionales:

DHCPv6 (RFC's: 3315, 3646, 3736),

IPsec (RFC's: 2404, 2410, 2451, 3602, 3566, 3686, 4301, 4303, 4305, 4312),

SIP (RFC's: 3261, 3264, 4566, 2617, 3665),

IPv6Enabled (que contiene dos sub-logos: WWW e ISP).

Fase 3: Su inicio estaba previsto para 2006, aunque en realidad aún no ha sido iniciado. La idea de esta fase, es que cubra los requisitos de la fase 2, más la obligatoriedad de cumplir las funcionalidades de IPSec.

Para cada una de las fases mencionadas existe una suite de tests de conformidad y de interoperabilidad.

Los aspectos técnicos del IPv6 Ready Logo Program son manejados por el IPv6 Logo Committee (v6LC) compuesto por: TAHI Project (Japón), UNH-IOL (U.S.A.), IRISA/INRIA (Francia), ETSI IPv6 Plugtest (Europa), TTA (Corea), BII (China) y NICI v6 Lab (Taiwan).

ETSI IPv6 Testing Project

Creada en Noviembre del 2004, su principal objetivo es reducir los costos del desarrollo y de los tests actuales, mediante el desarrollo de testing and test control notation versión 3 (TTCN-3), un lenguaje nuevo para crear tests de conformidad e interoperabilidad.

IPv6 Testing Tools

Mientras TTCN-3 aspira a suplantar los tests existentes, es posible encontrar varias suites de test para IPv6, casi todas ellas open-source.

TAHI, pionero en las test suites para IPv6, proporciona la herramienta V6eval, http://www.tahi.org/. La particularidad, por así llamarlo, de dicha suite, es que la misma está realizada para ejecutar en FreeBSD, por tanto es posible de ser ejecutada en sistemas *BSD.

UNH-IOL, http://www.iol.unh.edu/, proporciona un software propietario para testear la conformidad de IPv6.

El representante francés del IPv6 Ready Logo, IRISA http://www.irisa.fr/, es otro que proporciona la especificación v6LC y además, ha desarrollado por su cuenta tests para una variedad de tecnologías IPv6. Esta herramienta, cuenta con la virtud de tener un código de implementación cuya interpretación es relativamente accesible. Dado que el lenguaje utilizado es Perl, al ser un lenguaje interpretado, es posible el seguimiento de la ejecución paso a paso, así como el debugging de la misma, de una manera bastante amigable.

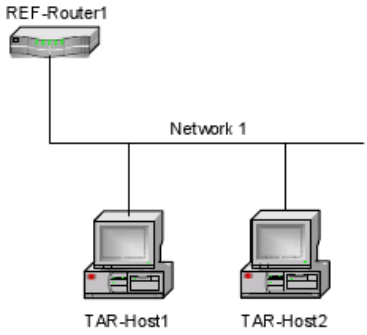
Testing IPv6 eventos y laboratorios

Como mencionamos anteriormente, para que un dispositivo obtenga el IPv6 Ready Logo, debe pasar los tests, interactuando con al menos dos hosts y dos routers de distintos fabricantes. La forma de poder obtener dicho logo es, enviando el dispositivo a un laboratorio de testeo, o participando de eventos de interoperabilidad para IPv6 que se realizan regularmente.

Para el caso de laboratorios de testeo, los mismos deben estar aprobados por v6LC. Al momento de la realización de este documento, los laboratorios existentes están distribuidos por distintas partes del mundo, (JATE) Japón, (CHT-TL) Taiwan, (TTA) Corea y (BII) China en Asia; (IRISA) Francia en Europa; y (UNH-IOL) U.S.A..

Test Setup: For each Part, connect hosts and routers as per the figures below. Allow time for all devices to perform stateless address autoconfiguration and Duplicate Address Detection.

Parts A-C:



TAR-Host1: Applicant Implementation
TAR-Host2: Vendor A/B Host
REF-Router1: any router

Procedure:

Part A: Link-Local unicast address (Host vs Host)

1. Transmit ICMPv6 Echo Requests from TAR-Host1 to the link-local address of TAR-Host2.
2. Observe the packets on Network1.
3. Transmit ICMPv6 Echo Requests from TAR-Host2 to the link-local address of TAR-Host1.
4. Observe the packets on Network1.

Figura 1: Topología y procedimiento para la ejecución del test de interoperabilidad 1.1.A

La operativa para realizar los testings, pasa por contar con la cantidad de dispositivos a testear, y un conjunto de routers que, dependiendo del test a verificar, permitan establecer distintas conexiones, para de esta manera poder generar las topologías de red requeridas. Dado que las topologías van cambiando a lo largo del test, se debe tener especial atención en las distintas conexiones a realizar según el test requerido. Esta tarea es para nada trivial, y ocasionalmente una considerable fuente de errores para la correcta ejecución de los distintos test.

El formato de los tests de interoperabilidad se divide en grupos de tests identificados por un número, y dentro de estos, la división es mediante letras. Los tests requieren una topología de red para cada uno de ellos, la organización del documento permite realizar más de una letra de test, si la topología requerida es la misma. El test de interoperabilidad 1.1.A, de acuerdo a su especificación original del documento del v6RL se presenta en la Figura 1. Los resultados esperados para dicho test, son apreciables en la Figura 2.

En este caso, se requiere armar una única subred conformada por un router y dos hosts. En esta subred se requiere validar que los dos host son capaces de comunicarse en ambos sentidos. Como se indica, luego de tener la red preparada, se deben conectar los dispositivos participantes y aguardar un tiempo, como para que estabilicen su configuración y sus valores de red, para luego transmitir un mensaje ICMP (ejecutar el comando ping) desde Host1 a Host2, y luego desde el Host2 al Host1.

En la parte resultados observados se detallan los mensajes (paquetes a nivel de red) que deben circular por la subred, para considerar el test como válido.

Observable Results:

- *Parts A and B*

  Step 2, 6: TAR-Host2 must receive all the ICMPv6 Echo Requests sent from TAR-Host1 and respond with ICMPv6 Echo Replies. The Source Address of the Echo Reply must be equal to the Destination Address that was in the Echo Request, and the Destination Address of the Echo Reply must be equal to the Source Address that was in the Echo Request.

  Step 4, 8: TAR-Host1 must receive all the ICMPv6 Echo Requests sent from TAR-Host2 and respond with ICMPv6 Echo Replies. The Source Address of the Echo Reply must be equal to the Destination Address that was in the Echo Request, and the Destination Address of the Echo Reply must be equal to the Source Address that was in the Echo Request.

  Possible Problems:

  - A passive node may not implement an application for sending Echo Requests.

Figura 2: Resultados y posibles problemas de la ejecución del test de interoperabilidad 1.1.A

2.2 Modelado del test de interoperabilidad en IPv6.

Procedemos inicialmente a buscar, investigar, analizar, fijando un objetivo, la obtención de un posible modelado con el que representar los casos indicados en el IPv6 Ready Logo.

Nuestra pauta inicial para la elaboración de los requerimientos del proyecto fue comenzar una búsqueda intentando determinar que avances existían en el área, y que caminos se estaban (o se habían) seguido.

Las búsquedas, fueron focalizadas mayoritariamente en lo que podíamos encontrar en Internet, dado que no se pudo encontrar bibliografía que hiciera referencia estricta al tema a buscar.

En base al tenor de la investigación, se decide centrar principalmente la búsqueda entre los documentos de conferencias y papers existentes que se encuentran alojados en la página de la IEEE (http://www.ieee.org), tratando de identificar si se hacía referencia a avances en el área, o se planteaban soluciones ya existentes.

Dado que no se pudo encontrar material que indicara novedades en esta área, se amplió el criterio de búsqueda, intentando encontrar referencias que se pudieran aplicar al modelado de test de interoperabilidad.

Así fue, que utilizando como criterios de búsqueda documentos que tuvieran dentro de sus palabras claves los términos "interoperability", "testing", "modeling", continuamos nuestra búsqueda dentro de la página IEEE.

Pese a la cantidad de documentos encontrados, contrariamente a una suposición inicial, la información encontrada no fue todo lo numerosa que se esperaba, por tanto, con los documentos recolectados, se procedió a realizar un análisis que permitiera llegar a una primera selección, en que a partir de la obtención de un modelado poder empezar a investigar la posible aplicación del mismo para nuestro propósitos.

Es así, que luego de un conjunto de lecturas de distintos documentos, se optó por pre-seleccionar cuatro posibles dentro de los mismos, los cuales podríamos llegar a utilizar como punto inicial, y en base al seleccionado poder determinar la viabilidad para la implementación del potencial modelado.

Es sobre estos documentos seleccionados que se realiza un pre-análisis, con el fin de poder interiorizarnos de su contenido, y determinar la viabilidad de escoger uno de ellos para la realización de nuestro modelado. Para facilitar la lectura de este documento, es que se incluye una descripción de dichos documentos en el Apéndice A.

2.3 Aplicación de técnicas de modelado existentes.

Luego de repasar el contenido de las distintas opciones, evaluando comprensión y viabilidad de la puesta en práctica de las mismas, y en el afán de poder proseguir con el desarrollo del modelado, es que, aunque las soluciones evaluadas, al menos a primera vista, no parecían satisfacer nuestras necesidades en su totalidad, decidimos escoger la que consideramos más apta, con la motivación de poder proseguir la elaboración de este proyecto.

2.3.1 Procedimiento inicial.

De las opciones seleccionadas y evaluadas, consideramos que la candidata, esto es, la más cercana a ser tenida en cuenta para desarrollar el modelado, es la que indicamos como CTIOA (Communicating Timed Input Output Automata) [6]. Dicha opción consiste en, definir un conjunto de autómatas de entrada y salida con temporizadores, y un conjunto de canales de comunicación, que serán los encargados de comunicar los autómatas entre sí. Se definen grafos que representan los distintos nodos componiendo los mismos en un grafo de alcance global.

En lo que respecta a las otras opciones, las indicadas como Intop [3] y U2TP (UML-2 Testing Profile) [4], no parecen brindar muchas posibilidades para obtener un pasaje automatizable desde la definición hasta la ejecución del mismo, es decir, no se pudo detectar de que manera, luego de realizado el modelado, se podía proseguir con la implementación.

Por otro lado, la opción CBR/CSP (Case-Based Reasoning/ Constraint Satisfaction Problem) [5], en principio, podría ser puesta en práctica, pero para llevarla a cabo, habría que recabar más información de como aplicar CBR, ya que al menos en los documentos revisados, no se encontró una manera de poner en práctica una automatización para aplicar los procesos de “conocimiento” de CBR.

Es de esta manera que en principio, la selección parece apuntar al menos y en base al espectro revisado, a la opción de aplicar CTIOA.

No parece haber impedimentos para aplicar esta metodología, sí, se torna necesario la inversión de tiempo para poder comprender la aplicación del algoritmo y los pasos a seguir. Una posible manera, que es la que llevaremos adelante, es pasar algunos de los casos del test de interoperabilidad de IPv6 a esta metodología, para poder determinar facilidades y dificultades de la aplicación de la misma.

2.3.2 Primeros pasos.

Con la potencial herramienta seleccionada, procedemos a intentar poder realizar el modelado.

Para comenzar centramos nuestros esfuerzos en la realización del CTIOA.

Es decir la tarea consistía en ir generando los autómatas para cada nodo (los TIOA), para luego ir componiendo los mismos y poder, finalmente, llegar a la obtención del CTIOA correspondiente al caso de prueba a representar.

En la tarea de obtención de los distintos autómatas, es donde se empezaron a presentar una serie de inconvenientes para poder representar los distintos casos de prueba del testing de interoperabilidad.

Por un lado, las máquinas de estado que potencialmente podíamos obtener, no eran tan simples como en principio esperábamos, así como la representación de algunas situaciones, como por ejemplo, el caso de cambiar una dirección IP de un Host (caso de prueba 1.2 de [1]), llevaba a modelados que, ademas de ser escasamente intuitivos, era de suponer que los mismos iban a contar con un número considerable de estados, lo que dificultaría no sólo la comprensión de los mismos, sino también la posible demostración que dichas máquinas de estado, eran la representación de los casos de prueba.

Podemos decir que, luego de estar un tiempo intentando aplicar esta metodología, se pudo comprender que para la obtención de resultados en un tiempo aproximado al de la duración de un proyecto de grado, no parecía ser la mejor idea la utilización de autómatas como forma de poder representar los casos de prueba de IPv6.

Igualmente, parece necesario aclarar, que la utilización de este método podría ser viable bajo otras condiciones, por ejemplo, destinando más recursos y tiempo a la elaboración del modelado, no deberíamos descartar el mismo como una opción a tomar en cuenta.

Un detalle a mencionar, no menos importante, ya que en realidad complicó aún más la tarea, es que pudiendo realizar una interpretación de los casos de prueba, los mismos están descritos de una manera casi lineal, por lo que intentar utilizar un autómata con timers, en muchos casos era agregarle una dificultad extra al modelado, ya que la virtud de la herramienta utilizada era convertir autómatas con timer en situaciones lineales, por lo que, y a modo de prueba, nos veíamos obligados casi a forzar los primeros casos de prueba en autómatas con timers, para convertirlos en programación lineal, cuando, a ciencia cierta, ya sabíamos el resultado de antemano.

Otro detalle a mencionar, es que el uso de la herramienta obligaba a representar los test mediante grafos. Esta operativa nos permitió poder identificar que cualquier modelado a utilizar deberá proveer alguna representación no solo de los pasajes de un nodo a otro, sino que deberá ser posible representar los nodos como entidades con propiedades asociadas, las cuáles pueden ser cambiadas durante la realización de los tests.

A continuación, y como muestra, ejemplificaremos utilizando el Test IP6Interop.1.1: ICMPv6 Echo Interoperability, parte A, para simplificar, sin la utilización de Timer.

El test es el reflejado en la Figura 1, y consiste en enviar un Request desde H1 a H2 (esperando su respuesta), para luego repetir el procedimiento en forma inversa, es decir enviar el Request desde H2 a H1.

La notación a utilizar, según el ejemplo, es el símbolo '!' para representar una salida, mientras que '?' representa una entrada.

En el primer recuadro identificamos el proceso que involucra a H1, se envía una solicitud, se recibe respuesta, se recibe solicitud, se envía respuesta. El segundo recuadro es la misma situación, vista desde H2.

El tercer recuadro es el producto que se obtiene al componer los dos grafos. Pese a que el test es relativamente sencillo, la aplicación de este modelado no parece clarificar la representación. El resultado mostrado no permite intuir de manera fácil que lo obtenido es representación de un test de interoperabilidad.

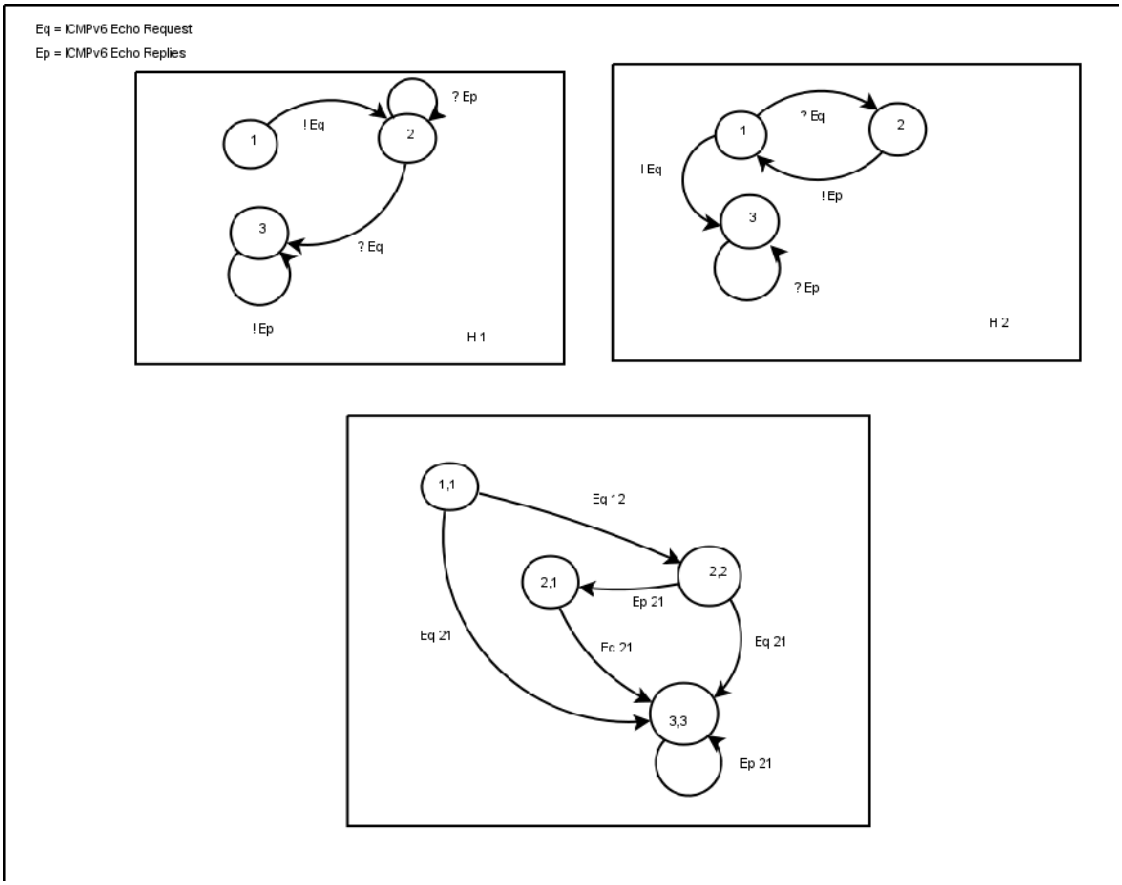


Figura 3: Aplicación de CTIOA para el test 1.1.A

En vista de los resultados obtenidos, es que debemos replantear las distintas posibilidades con las que contamos hasta ahora. Por un lado podemos considerar alguna de las opciones ya descartadas, investigando nuevamente, intentado detectar si es que se nos paso algo. Otra forma sería buscar alguna otra opción viable, volviendo a investigar en Internet, para que de esta manera el proyecto pueda ser re-encaminado. Estas opciones, no parecen ser lo suficientemente alentadoras, como para invertir más tiempo en ellas.

Las opciones descartadas, lo fueron porque realmente no satisfacían nuestras necesidades, y dado que elegimos la que en principio pensamos era la más óptima, resulta bastante desalentador, y difícilmente esperable, suponer que si no pudimos llegar utilizando la mejor opción, sí lo vayamos a hacer eligiendo la menos mala, por más que le tildemos con optimismo de suponer que es la segunda más buena.

Por otro lado, las búsquedas realizadas en Internet, se hicieron a conciencia, intentando encontrar material que pudiera aportar a nuestra realización, por lo que es bastante difícil suponer, que en tan poco tiempo, surja material que antes no estaba.

Es decir, en cuanto a la búsqueda de material en el área, no fue posible encontrar herramientas o procesos aplicables a un test de interoperabilidad, capaces de ser utilizados en el contexto de este trabajo.

Por decirlo de alguna manera, en este momento se estaba en una encrucijada, respecto a poder definir de que manera seguir con la elaboración de este proyecto.

Luego de buscar y manejar distintas opciones, intentamos evaluar una nueva posibilidad. La misma consiste en, cambiar el foco desde donde comenzar nuestra tarea, y replantear la búsqueda de una solución.

3. Propuesta de modelado.

El enfoque consiste en partir desde los casos de prueba, considerarlos como el punto de entrada, y a partir de ellos, elaborar un modelado.

Es decir, intentaremos representar los casos de prueba del IPv6 Ready Logo, realizando una interpretación de los mismos a partir del documento donde se describe el test de interoperabilidad, y considerando poder representarlos de forma abstracta. Dados los distintos tests, buscamos que entidades y operaciones debemos representar y procedemos a determinar una representación de los mismos.

Es decir nuestro modelado, parte del standard (la descripción del test de interoperabilidad de IPv6 Ready Logo), toma directamente de ahí una serie de enunciados que los define como axiomas, y en base a los mismos, propone teoremas que aplican sobre el documento.

Esta aproximación, se realiza desde un punto no tan tradicional en comparación con las analizadas anteriormente, y podríamos ejemplificarla de la siguiente manera:

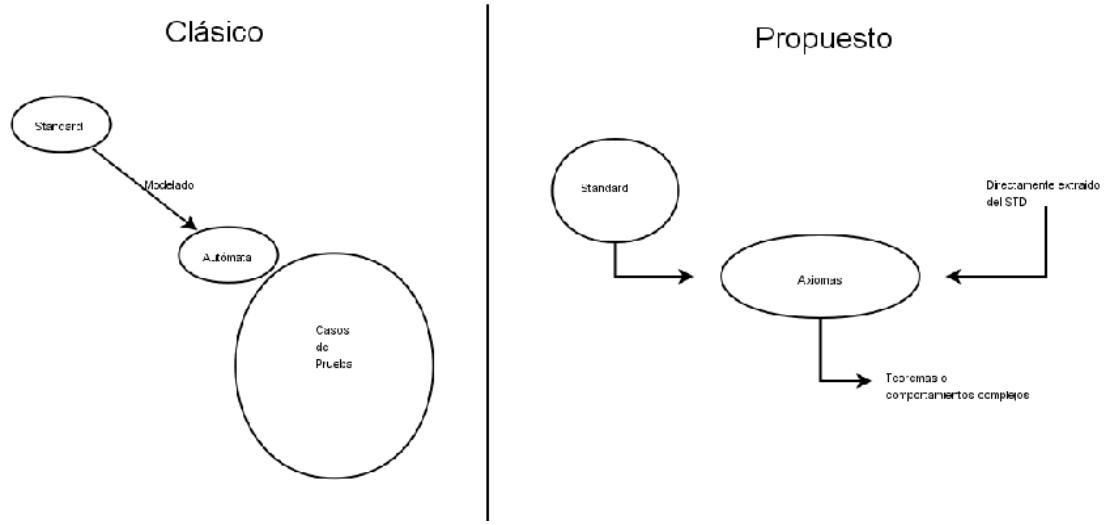


Figura 4: Modelado propuesto

La posible solución que podemos llegar a implementar, al menos en nuestro análisis inicial, tiene la dificultad que no existe un fundamento teórico sobre el cuál apoyarse en el desarrollo, pero a su vez, la dificultad (quizás, y desde una óptica positivista le podríamos llamar reto) se puede ver como una ventaja, ya que brinda la oportunidad de poder tomarnos ciertas libertades en la realización de nuestro modelado, dado que mientras podamos definir un modelo que sea consistente, estaríamos definiendo una posible solución.

Como primer acercamiento, se parte del supuesto, por simplificación, que un test de interoperabilidad, es posible verlo como una serie de pruebas para determinar si un par de nodos (Hosts y/o Routers) se comunican correctamente, utilizando los mensajes requeridos de acuerdo a la descripción del test.

Los distintos casos de prueba detallados en el test de interoperabilidad, son para verificar la existencia de dicha comunicación, la cuál principalmente consiste en el envío de Echo Requests, para la posterior observación de la existencia de respuestas a

los mismos, Echo Replies, en distintas topologías de red.

La existencia de las distintas topologías, en el conjunto de tests, tienen la intención de cubrir un conjunto de casos, para de esta manera afirmar que si los dispositivos se comunican exitosamente en éstos, deberían comunicarse correctamente en cualquier configuración de red.

Es así, que podemos considerar a cada caso de prueba, como una topología de red, con un conjunto de dispositivos (Host y/o Router) sobre los cuales se debe verificar si los dispositivos a testear son capaces de comunicarse entre sí.

Los Router, serán los encargados de comunicar las distintas subredes requeridas para los casos a testear, razón por la cuál podrán tener más de una conexión de red, siendo cada conexión aplicada a una tarjeta de red.

Una subred en este contexto por tanto, consiste en una porción de la red, donde cada host existente tiene acceso directo a otro host, sin la necesidad de tener que pasar por un Router para poder intercambiar mensajes.

Dicho de otra manera, y en el afán de empezar una formalización a lo mencionado anteriormente, es que podemos afirmar, en forma genérica, que un caso de prueba T, contendrá dos conjuntos: un conjunto de hosts H, y otro de routers R, es decir:

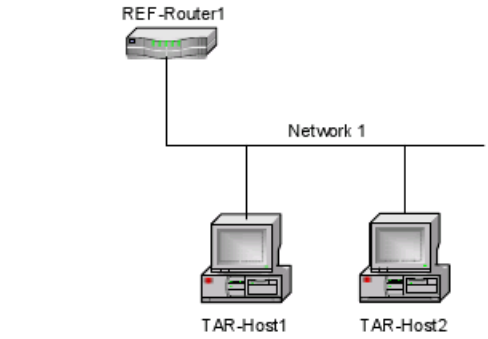
```
T = {X, R},
donde, por ejemplo
X = {H1, H2, H3}, con
H1 = {pc1, pc2, pc3}, etc y
R = {R1, R2}, con,
por ejemplo R1 = {r(H1,H2)}, como representación de un
router que comunica la subred 1 con la subred 2.
```

También, y en base al contenido de los tests, identificamos las interfaces de cada router, es decir, para, por ejemplo, R1 = {r(H1,H2)}, podemos considerar que R1, tiene dos interfaces, la que involucra la conexión con H1 y la de H2. A los efectos de nuestro modelado, las interfaces van a tener cierta importancia, pero se debe tener en cuenta que la existencia de las mismas esta directamente vinculada a la existencia de un router.

Volviendo al ejemplo visto previamente:

Test Setup: For each Part, connect hosts and routers as per the figures below. Allow time for all devices to perform stateless address autoconfiguration and Duplicate Address Detection.

Parts A-C:



TAR-Host1: Applicant Implementation
TAR-Host2: Vendor A/B Host
REF-Router1: any router

Figura 5: Test de interoperabilidad 1.1.A

Podemos modelar la topología del caso de prueba como:

```
T1={X,R}
con X = {H1},
    H1={pc1,pc2},
    pc1=TAR-Host1
    pc2= TAR-Host2
    R={R1} ,
es decir R1= REF-Router1, donde R1={r(H1,H1)}
```

Por tanto, podemos considerar, que es posible afirmar que los casos de prueba, básicamente involucran la participación de distintas sub-redes conteniendo hosts, conectadas entre sí por routers.

Claro está, que aún resta definir las distintas operaciones que podemos aplicar sobre los nodos, para establecer distintas interacciones que se hacen necesario testear.

Determinando esta definición de grafos, pasamos a detallar las distintas hipótesis que podemos indicar sobre la definición.

Inicialmente, detallaremos las mismas utilizando la empiria y los conocimientos previos con que contamos de la intercomunicación de redes.

Estas afirmaciones, pese a que en este momento no realizaremos una demostración formal, son fácilmente comprobables (o al menos esperamos que así sea), para personas con conocimientos de redes computacionales.

Estas hipótesis de trabajo, sobre las cuales trabajaremos, son las que nos permitirán avanzar en un modelado que sea capaz de representar la suite del test de interoperabilidad.

Las hipótesis a enunciar, más las operaciones a definir, nos permitirán representar la suite de los casos de prueba de la fase uno y dos.

3.1 Conjunto de hipótesis

A continuación, enunciaremos un conjunto de hipótesis sobre las cuáles realizaremos los diferentes casos de tests a utilizar.

Para simplificar nuestra representación, utilizaremos el símbolo ⇄ con la finalidad de indicar que dos entidades "se comunican", esto es frente a una solicitud de comunicación de un nodo A, el nodo B es capaz a responder.

Formalizando un poco más, A ⇄ B significa que el nodo B responde con un Echo Reply a la solicitud de Echo Request enviada por A y recibida por B.

Por oposición, utilizaremos el símbolo ⇊ con la finalidad de indicar que dos entidades "NO se comunican", es decir el nodo A solicita comunicarse con B, y B no responde.

Hipótesis 1:

Dado T={X,R}, con X={H1} y R = {R1}, donde R1={r(H1,H1)}

∀ pc1, pc2 ∈ H1, se cumple: pc1 ⇄ pc2 y pc2 ⇄ pc1.

Es decir para dos hosts de una sub-red, si A se comunica con B, B se comunica con A.

Hipótesis 2:

Dado T={X,R}, con X={H1,H2} y R = {R1}, donde R1={r(H1,H2)}

∀ pc1 ∈ H1, pc2 ∈ H2, se cumple: si ∃ R1 / R1={r(H1,H2)} entonces R1 ⇄ pc1 y R1 ⇄ pc2.

Para dos hosts de distintas sub-redes, si existe un router que comunica ambas sub-redes, entonces el router se comunica con los hosts de ambas sub-redes.

Hipótesis 3:

Dado T={X,R}, con X={H1,H2} y R = {R1}, donde R1={r(H1,H2)}

∀ pc1 ∈ H1, pc2 ∈ H2, se cumple: si ∃ R1 / R1={r(H1,H2)} entonces pc1 ⇄ pc2.

Se pude considerar como un corolario del anterior, si un router se comunica con dos sub-redes, entonces los hosts de la sub-red 1, se pueden comunicar con los de la sub-red 2.

Hipótesis 4:

Para este caso asumamos X={H1,...} y R={R1,R2,...}, con R1={r(Hi,Hj)} y R2={r(Hk,Hl)}

∀ R1 ∈ R, R2 ∈ R, se cumple: si i==k o i==l o j==k o j==l entonces R1 ⇄ R2 (por supuesto que también se cumple R2 ⇄ R1).

Es decir si dos routers, que comunican sub-redes, tienen una sub-red común, entonces ambos routers se comunican entre sí.

Hipótesis 5:

Esta propiedad la podemos considerar como una extensión de la anterior:

Para este caso asumamos X={H1,...,H(n)} y R={R1,R2,...,R(n-1)},

con R1={r(H1,H2)},..., R(n-1)={r(H(n-1),H(n))},

o dicho de forma genérica, Ri={r(Hi,Hi+1)}, con i=1..(n-1).

∀ pc1 ∈ Hi, pcn ∈ Hj, se cumple: pc1 ⇄ pcn.

Es decir, si tenemos routers que unen todas las sub-redes de un sistema, entonces todos los hosts de las distintas sub-redes, se comunican entre sí.

3.2 Operaciones

A continuación, indicamos las operaciones que se pueden realizar sobre los distintos componentes de un caso de prueba.

Ŵ

Ŵ (o Wait), es una función que aplica sobre un test, y su ejecución consiste en esperar, ya sea una cantidad de tiempo necesaria y a determinar, o la espera de algún evento indicado. Básicamente, esta operación será utilizada para luego de un cambio de configuración por la ejecución de la función Ď y/o Ě (es decir la modificación de algún atributo de algún nodo).

Para la implementación práctica de dicha función, se incluirá un par de parámetros en su ejecución, el primero representará una condición, por la cual se esperará a que se cumpla la misma para dar por terminada la espera, el segundo se referirá a una cantidad de tiempo (en segundos) para el caso que no se cumpla la condición y evitar una espera infinita. Para el caso en que no se utilice el primer parámetro y sólo se utilice el segundo, la función se comportará como una espera de “X” segundos, siendo “X” el parámetro. Para el caso en que se utilice el primer parámetro se incluye el “comodín” Enter, en ese caso se esperará hasta que el ejecutante del test presione Enter.

Ŝ

Ŝ (o Setting) es una función que aplica sobre un componente (host, router o interface), y el cometido es cambiar alguna propiedad que tenga asignada por defecto. En particular, podemos asumir que para cada nodo que representamos como pc1, esto no es más que una abreviación de lo que en definitiva se representa, es decir $pc1_{pc1}$, o sea que un nodo tiene un conjunto de propiedades que lo identifica del resto de los nodos del caso de prueba.

En particular y a efectos de los casos de prueba 1.2, podemos utilizar la función Ŝ con el fin de cambiar una dirección por defecto. Para este caso, la invocación de dicha función es la siguiente:

$\hat{S}^{A}(B,C) = B_C$, con B y C nodos, o sea, se le asigna al nodo B, la propiedad del nodo C, que en este caso, dado que el supra-índice es A (address), es la dirección.

Los posibles supra-índices, se señalan a continuación, con una breve descripción de lo que hacen. La invocación de dicha función es la siguiente:

A (address) : cambia el valor de la dirección. Se le pasan dos parámetros, el primero indica sobre el nodo que aplica, y el segundo, la dirección del nodo a asignar.

R (router advertisement): configuración de RA. La invocación a esta funcionalidad aplica sobre un enlace, por tanto, recibe dos parámetros de entrada, con los que se identifica un enlace. El primero debe ser un router, y el segundo una sub-red.

Los siguientes son hasta siete parámetros a asignar:

1. el primero indica el prefijo a asignar,
2. el segundo el tiempo de vida,
3. el tercero el intervalo,
4. el cuarto el destino de un Redirect message,
5. el quinto el valor de MTU de ese enlace,
6. el sexto la indicación del Router elegido como mejor primer salto (better first hop),
7. el séptimo, la configuración de un enrutado estático, indicando el camino para el próximo salto, para ir a una cierta sub-red.

La ausencia de alguno de esos parámetros implica que la configuración para ese valor será la que se provee por defecto.

Hasta el momento y para las operaciones previas hemos utilizado todos los componentes en el formato “abreviado”, es decir, por ejemplo pc1, que como dijimos anteriormente, es $pc1_{pc1}$. Otra forma equivalente de ver lo anterior, es apreciando la siguiente igualdad:

$\hat{S}^{A}(pc1,pc1) = pc1_{pc1} = pc1$, siendo A la operación correspondiente al cambio de dirección para ese nodo.

Operaciones:

i. Caso 1

$\forall$ pc1, pc2, pc3 $\in$ H1, se cumple:

$pc1_{pc1} \rightleftarrows pc2_{pc2}$

$pc1_{pc1} \rightleftarrows pc3_{pc3}$

$pc2_{pc2} \rightleftarrows pc3_{pc3}$

Si ejecutamos, $\hat{S}^{A}(pc1,pc2) = pc1^{A}_{pc2}$.

Se cumple:

$pc1^{A}_{pc2} \Downarrow pc2_{pc2}$

$pc1^{A}_{pc2} \Downarrow pc3_{pc3}$

$pc2_{pc2} \rightleftarrows pc3_{pc3}$

Esto ocurre, porque al pc1 tratar de asignarse la dirección de pc2, la misma ya la tiene asignada pc2, por lo que el nodo no es capaz de levantar con dicha dirección.

ii. Caso 2

∀ pc1, pc2, pc3 tal que pc1, pc2 ∈ H1 y pc3 ∈ H2. Si existe R1={r(H1,H2)}, se cumple:

$pc1_{pc1} \rightleftarrows pc2_{pc2}$

$pc1_{pc1} \rightleftarrows pc3_{pc3}$ (porque: $pc1_{pc1} \rightleftarrows$ R1 y R1 $\rightleftarrows pc3_{pc3}$).

$pc2_{pc2} \rightleftarrows pc3_{pc3}$ (porque: $pc2_{pc2} \rightleftarrows$ R1 y R1 $\rightleftarrows pc3_{pc3}$).

Si ejecutamos, $\hat{S}^{A}(pc1,pc3) = pc1^{A}_{pc3}$.

Se cumple:

$pc1^{A}_{pc3} \Downarrow pc2_{pc2}$

$pc1^{A}_{pc3} \Downarrow pc3_{pc3}$

$pc2_{pc2} \rightleftarrows pc3_{pc3}$

Igual razón que para el punto anterior.

Existe una extensión a la función anterior, que llamaremos $\check{T}^{A}$, la ejecución de esta función varía en que el segundo parámetro es una sub-red (H1, H2, etc), y la ejecución significa asignarle al nodo una dirección correspondiente a la sub-red asignada como parámetro, con la condición que la misma no se corresponda a la de un nodo ya existente para esa sub-red (claramente si se deseara hacer coincidir, sería válida la aplicación de la función previa). Podemos ver la notación de la siguiente manera:

Sea H1=(pc1) y H2=(pc2,pc3).

$\check{T}^{A}(pc1,H2) = pc1^{A}_{H2}$, y la dirección a asignar a pc1 difiere de las direcciones por defecto pc2, pc3.

Propiedades

∀ pc1, pc2, pc3 tal que pc1, pc2 ∈ H1 y pc3 ∈ H2. Si existe R1={r(H1,H2)}, se cumple:

$pc1_{pc1} \rightleftarrows pc2_{pc2}$

$pc1_{pc1} \rightleftarrows pc3_{pc3}$ (porque: $pc1_{pc1} \rightleftarrows$ R1 y R1 $\rightleftarrows pc3_{pc3}$).

$pc2_{pc2} \rightleftarrows pc3_{pc3}$ (porque: $pc2_{pc2} \rightleftarrows$ R1 y R1 $\rightleftarrows pc3_{pc3}$).

Si ejecutamos, $\check{T}^{A}(pc1,H2) = pc1^{A}_{H2}$.

Se cumple:

$pc1^{A}_{H2} \Downarrow pc2_{pc2}$

$pc1^{A}_{H2} \Downarrow pc3_{pc3}$

$pc2_{pc2} \rightleftarrows pc3_{pc3}$

No es posible que este nodo pueda tener el prefijo de dirección, con un valor igual al de otra sub-red ya definida.

Ď

Ď (disable) es una función que aplica sobre un conjunto de nodos, y su ejecución consiste en deshabilitar la conectividad de dicho nodo de su sub-red (en principio, esto es posible, al desconectar las interfaces del nodo de la red).

Para la representación de la aplicación de la siguiente operación, incluimos la siguiente nomenclatura: Ď(pc1) es representable mediante (~~pc1~~), y para este caso, se cumple:

Hipótesis 6:

pc1 ⇄ pc2 y pc2 ⇄ pc1, si se aplica Ď(pc1) = Ŵ() . (~~pc1~~), se deduce:

~~pc1~~ ⇊ pc2 y pc2 ⇊ ~~pc1~~.

Corolario

Además, se denota:

Ď(pc1) = Ŵ() . (~~pc1~~) y Ď(~~pc1~~) = (~~pc1~~).

El parámetro de la función, puede ser más de un nodo, en ese caso, por convención, se ejecutará de izquierda a derecha, es decir:

Ď(pc1,pc2,pc3) = Ď(pc1) . Ď(pc2) . Ď(pc3)

Ě

Ě (enabled) es una función que aplica sobre un conjunto de nodos, y la ejecución consiste en habilitar la conectividad de dicho nodo en su sub-red.

Claramente, podemos identificar la función como la operación inversa a Ď.

Por tanto, tenemos la siguiente propiedad:

Ě(~~pc1~~) = Ŵ . (pc1) y Ě(pc1) = (pc1).

De igual manera que para Ď, el parámetro de la función, puede ser más de un nodo, en ese caso, por convención, se ejecutará de izquierda a derecha, es decir:

Ě(pc1,pc2,pc3) = Ě(pc1) . Ě(pc2) . Ě(pc3)

Además, es claro que:

Ě(Ď(pc1)) = Ŵ() . Ŵ() . pc1.

y por “inverso”

Ď(Ě(~~pc1~~)) = Ŵ() . Ŵ() . ~~pc1~~

3.3 Modelado de la suite de tests de interoperabilidad.

Con las herramientas definidas en el capítulo anterior, procedemos a representar los distintos test de interoperabilidad del v6RL (IPv6 Ready Logo).

Para facilitar la descripción de los distintos casos, denotamos lo siguiente:

$\rightleftarrows$: significa que el Echo Request es a la dirección link-local.

$\rightleftarrows_{G}$: significa que el Echo Request es a la dirección Global unicast. Si existiera más de un prefijo se indicará como $_{Gi}$ el Echo Request asociado al prefijo "$_{i}$".

$\rightleftarrows_{A}$: significa que el Echo Request es a la dirección All Nodes multicast.

$\rightleftarrows_{AR}$: significa que el Echo Request es a la dirección All Router multicast.

$\rightleftarrows_{B<NUMERO>}$: significa transmitir un Echo Request de tamaño igual a <NUMERO> de bytes.

Claro está, que la representación del envío de un Echo Request, para el cuál NO se recibe un Echo Replies, es $\Downarrow$.

A continuación detallamos la representación de los distintos casos de prueba utilizando la notación indicada previamente.

Test 1.1.

En este grupo de tests se verifica la comunicación bidireccional entre dos nodos, es decir:

dadas diversas topologías de red, se cumple que: dados pc1, pc2, si pc1 ⇄ pc2 entonces pc2 ⇄ pc1.

Parte A-C. T1.1.A-C={H1,R1}, con H1={pc1,pc2} y R1 = {r(H1,H1)}. Para estos casos se considera una única subred conformada por un router y dos hosts. Se verificara la comunicación entre los dos hosts, mediante solicitudes a la dirección local (A), global (B) y multicast C.

Verificar:

A:

pc1 ⇄ pc2

pc2 ⇄ pc1

B:

pc1 $\rightleftarrows_G$ pc2

pc2 $\rightleftarrows_G$ pc1

C:

Ď(R1)

pc1 $\rightleftarrows_A$ pc2

pc2 $\rightleftarrows_A$ pc1

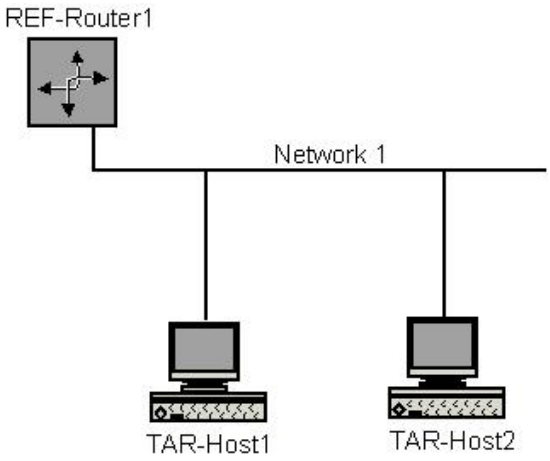


Figura 6: Topología test 1.1.A-C

Parte D-F. T1.1.D-F={H1,R1}, con H1={pc1} y R1 = {r(H1,H1)}. Para estos casos se considera una única subred conformada por un router y un host. Se verificara la comunicación entre ambos dispositivos, mediante solicitudes a la dirección local (A), global (B) y multicast C.

Verificar:

D:

pc1 ⇄ R1

R1 ⇄ pc1

E:

pc1 $\rightleftarrows_G$ R1

R1 $\rightleftarrows_G$ pc1

F:

pc1 $\rightleftarrows_A$ R1

R1 $\rightleftarrows_A$ pc1

pc1 $\rightleftarrows_{AR}$ R1

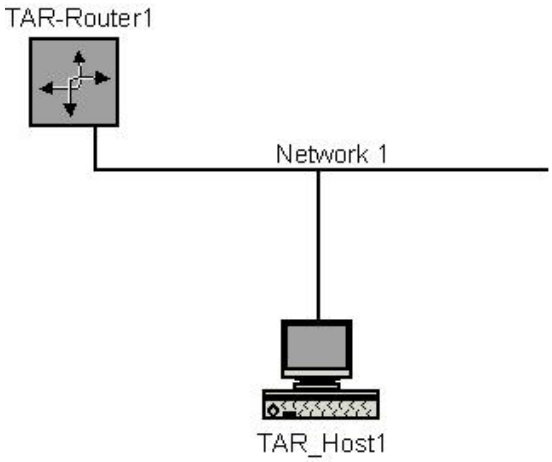


Figura 7: Topología test 1.1.D-F

Parte G-I. T1.1.G-I={H1,R1,R2}, con H1={}, R1={r(H1,H1)} y R2={r(H1,H1)}. Para estos casos se considera una única subred conformada por dos routers. Se verificara la comunicación entre ambos dispositivos, mediante solicitudes a la dirección local (A), global (B) y multicast C.

Verificar:

G:

R1 ⇄ R2

R2 ⇄ R1

H:

R1 $\rightleftarrows_G$ R2

R2 $\rightleftarrows_G$ R1

I:

R1 $\rightleftarrows_A$ R2

R2 $\rightleftarrows_A$ R1

R1 $\rightleftarrows_{AR}$ R2

R2 $\rightleftarrows_{AR}$ R1

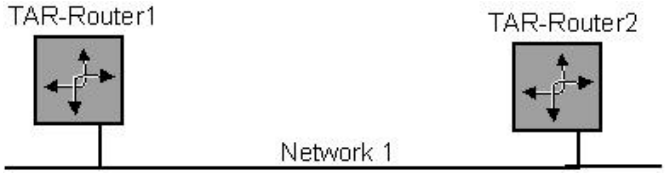


Figura 8: Topología test 1.1.D-G

Test 1.2

En este grupo de tests se verifica la autoconfiguración, y la detección de direcciones duplicadas para dos dispositivos.

Parte A-B. T1.2.A-B={H1,R1}, con H1={pc1,pc2} y R1 = {r(H1,H1)}. Para estos casos se considera una única subred conformada por un router y dos hosts. Se verificara la detección de direcciones duplicadas, para el caso de direcciones únicas (A) y las de ambos hosts duplicadas (B) .

Verificar:

A:

Ď(R1,pc1,pc2)

Ě(R1,pc1,pc2)

Ŵ

R1 ⇄ pc1

R1 ⇄ pc2

Ď(R1,pc1,pc2)

Ě(R1,pc2,pc1)

Ŵ

R1 ⇄ pc1

R1 ⇄ pc2

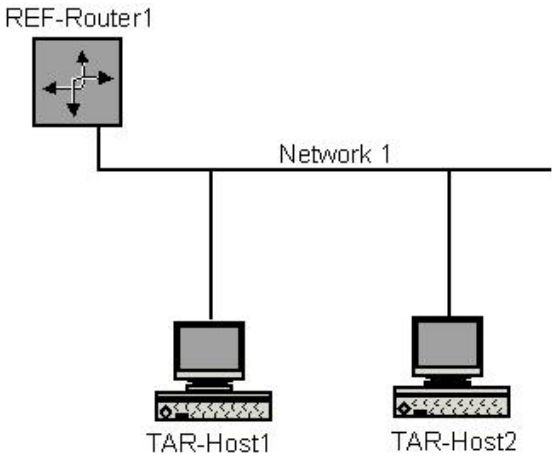


Figura 9: Topología test 1.2.A-B

B:

Ď(R1,pc1,pc2)

$\hat{S}^A$(pc2,pc1)

Ě(R1,pc2,pc1)

Ŵ

R1 ⇄ pc2

R1 ⇊ pc1

Ď(R1,pc1,pc2)

$\hat{S}^A$(pc2,pc2)

$\hat{S}^A$(pc1,pc2)

Ě(R1,pc1,pc2)

Ŵ

R1 ⇄ pc1

R1 ⇊ pc2

Parte C-D. T1.2.C-D={H1,R1}, con H1={pc1,pc2} y R1 = {r(H1,H1)}. Para estos casos se considera una única subred conformada por un router y dos hosts. Se verificara la detección de direcciones duplicadas, para el caso de direcciones únicas (A) y duplicadas entre un host y el router (B).

Verificar:

C:

Ď(R1,pc1,pc2)

Ě(pc1,R1,pc2)

Ŵ

pc2 ⇄ pc1

pc2 ⇄ R1

Ď(R1,pc1,pc2)

Ě(R1,pc1,pc2)

Ŵ

pc2 ⇄ pc1

pc2 ⇄ R1

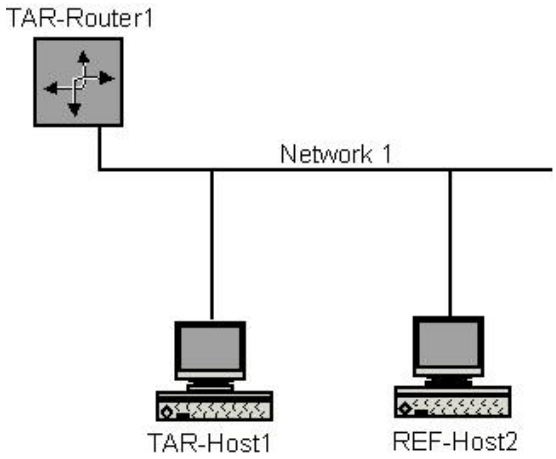


Figura 10: Topología test 1.2.C-D

D:

Ď(R1,pc1,pc2)

$\hat{S}^{A}$(R1,pc1)

Ě(R1,pc1,pc2)

Ŵ

pc2 ⇄ R1

pc2 ⇊ pc1

Ď(R1,pc1,pc2)

$\hat{S}^{A}$(R1,R1)

$\hat{S}^{A}$(pc1,R1)

Ě(pc1,R1,pc2)

Ŵ

pc2 ⇄ pc1

pc2 ⇊ R1

Parte E-F. T1.1.E-F={H1,R1,R2}, con H1={pc1}, R1={r(H1,H1)} y R2={r(H1,H1)}. Para estos casos se considera una única subred conformada por dos routers y un host. Se verificara la detección de direcciones duplicadas, para el caso de direcciones únicas (A) y las de ambos routers duplicados (B).

Verificar:

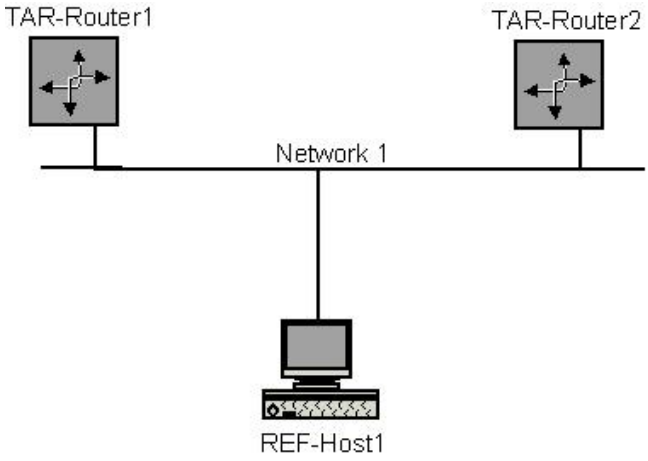


Figura 11: Topología test 1.2.E-F

E:

Ď(R1,pc1,R2)

Ě(R1,pc1,R2)

Ŵ

pc1 ⇄ R1

pc1 ⇄ R2

Ď(R1,pc1,R2)

Ě(R2,pc1,R1)

Ŵ

pc1 ⇄ R1

pc1 ⇄ R2

F:

Ď(R1,pc1,R2)

$\hat{S}^A$(R2,R1)

Ě(pc1,R2,R1)

Ŵ

pc1 ⇄ R2

pc1 ⇊ R1

Ď(R1,pc1,R2)

$\hat{S}^A$(R2,R2)

$\hat{S}^A$(R1,R2)

Ě(pc1,R1,R2)

Ŵ

pc1 ⇄ R1

pc1 ⇊ R2

Test 1.3

En este grupo de tests se verifica que un dispositivo puede realizar correctamente el descubrimiento de los prefijos de los router advertisment.

Parte A-C. T1.3.A-C={H1,R1}, con H1={pc1,pc2} y R1 = {r(H1,H2)}. Para estos casos se considera una única subred conformada por un router y dos hosts. Se verifica el caso de un prefijo solo (A), de múltiples prefijos (B), y de prefijos con tiempo de vida (C).

Verificar:

A:

$\hat{W}$

pc2 $\rightleftarrows_G$ pc1

B:

$\hat{S}^R$(R1,H1,prefix1,(>0),,,,,)

$\hat{S}^R$(R1,H1,prefix2,(>0),,,,,)

$\hat{W}$

pc2 $\rightleftarrows_{G1}$ pc1

pc2 $\rightleftarrows_{G2}$ pc1

C:

$\hat{S}^R$(R1,H1,prefix1,30,,,,,)

$\hat{W}$

pc2 $\rightleftarrows_G$ pc1

$\hat{W}$(35)

pc1 $\rightleftarrows_G$ pc2

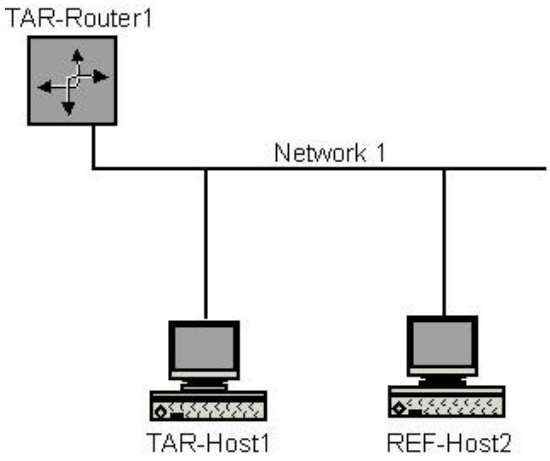


Figura 12: Topología test 1.3

Test 1.4

En este grupo de tests se verifica que un dispositivo puede procesar correctamente los router discovery.

Parte A-B. T1.4.A-B={H1,R1,H2}, con H1={pc1}, H2={pc2} y R1 = {r(H1,H2)}. Para estos casos se consideran dos subredres de un host cada una conectadas por un router. Se verifica el caso de un tiempo de vida distinto a cero (A), y de tiempo de vida igual a cero (B).

Verificar:

A:

$\hat{S}^R$(R1,H1,,600,60,,,,)

$\hat{S}^R$(R1,H2,,600,60,,,,)

pc2 $\rightleftarrows_G$ pc1

B:

$\hat{S}^R$(R1,H1,,0,normal,,,,)

$\hat{S}^R$(R1,H2,,>normal,,,,,)

pc2 $\rightleftarrows_G$ pc1

$\hat{S}^R$(R1,H1,,600,60,900,,,)

$\hat{S}^R$(R1,H2,,600,60,900,,,)

pc2 $\rightleftarrows_G$ pc1

$\hat{S}^R$(R1,H1,,0,60,,,,)

pc2 $\rightleftarrows_G$ pc1

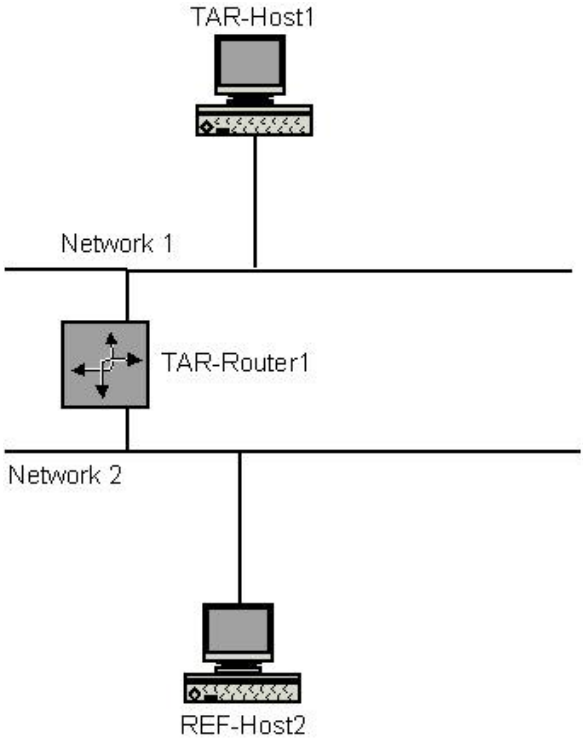


Figura 13: Topología test 1.4

Test 1.5

En este test se verifica que un dispositivo puede procesar correctamente funciones de redireccionamiento (better first-hop).

T1.5={H1,R1,H2,R2}, con H1={pc1}, H2={pc2}, R1 = {r(H1,H1)} y R2 = {r(H1,H2)}. Para este caso se consideran dos subredres de un host cada una conectadas por un router, y otro router que forma parte de una de las subredes.

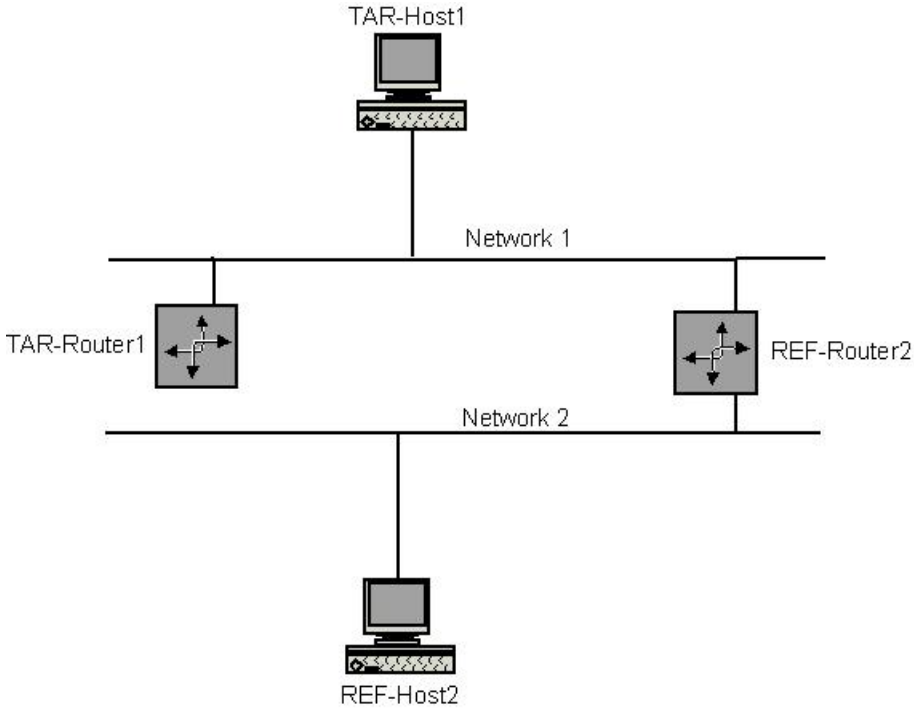


Figura 14: Topología test 1.5

Verificar:

A:

pc2 $\rightleftarrows_G$ pc1

$\hat{S}^R$(R1,H1,,,,,,R2,) //Así represento el better first hop

pc2 $\rightleftarrows_G$ pc1

Test 1.6

En este test se verifica que un dispositivo puede manejar los path MTU (Maximum Transmission Unit) discovery y la fragmentación.

Las casos A y B verifican PMTU Discovery y los casos C, D y E verifican fragmentación y rearmado.

Parte A. T1.6.A={H1,R1,H2}, con H1={pc1}, H2={pc2} y R1 = {r(H1,H2)}. Para este caso se consideran dos subredres de un host cada una conectadas por un router. Se debe verificar el correcto funcionamiento cuando ambas subredes tienen diferente MTU.

Verificar:

$\hat{S}^R$(R1,H1,,,,,1500,,)

$\hat{S}^R$(R1,H2,,,,,1280,,)

pc2 $\rightleftarrows_{GB1500}$ pc1 // transmitir 1500 bytes

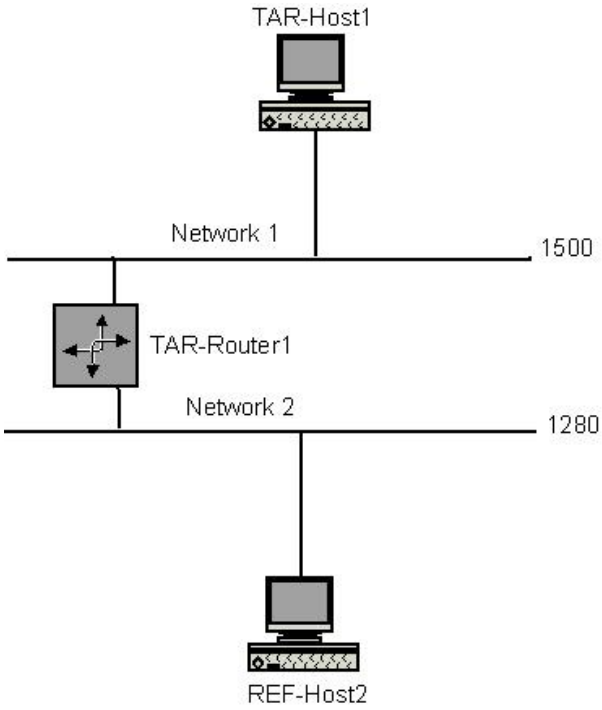


Figura 15: Topología test 1.6.A

Parte B. Para estos casos se consideran dos subredres conectadas por un router, en una de ellas se encuentra un router y en la otra un host. Se debe verificar el correcto funcionamiento cuando ambas subredes tienen diferente MTU. Ambas partes (B.1 y B.2) se realizan intercambiando la ubicación de ambos routers.

Parte B.1. T1.6.B={H1,R1,R2}, con H1={pc1}, R1 = {r(R2,R2)} y R2 = {r(H1,R1)}.

Verificar:

$\hat{S}^R$(R1,R2,,,,,1500,,)

$\hat{S}^R$(R2,H1,,,,,1280,,)

pc1 $\rightleftarrows_{GB1500}$ R1

Parte B.2. T1.6.B={H1,R1,R2}, con H1={pc1}, R2 = {r(R1,R1)} y R1 = {r(H1,R2)}

$\hat{S}^R$(R2,R1,,,,,1500,,)

$\hat{S}^R$(R1,H1,,,,,1280,,)

pc1 $\rightleftarrows_{GB1500}$ R1

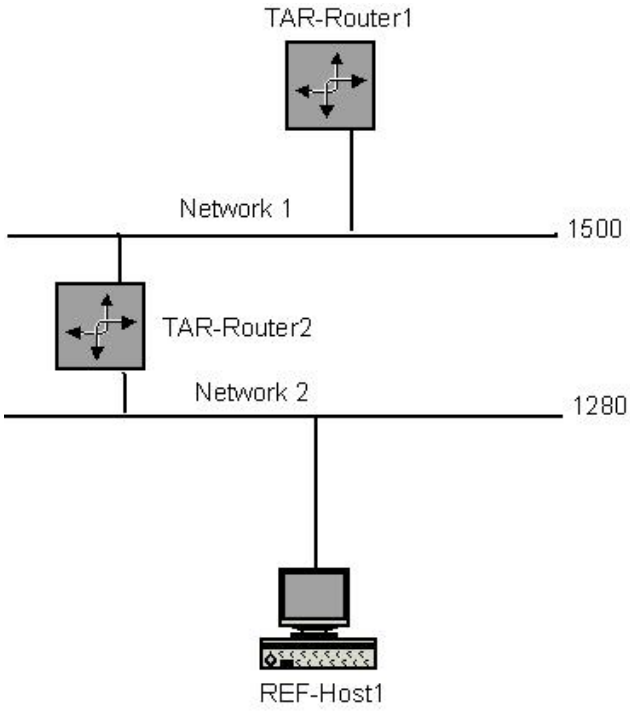


Figura 16: Topología test 1.6.B

Parte C. T1.6.C={H1,R1,H2}, con H1={pc1}, H2={pc2} y R1 = {r(H1,H2)}. Para este caso se considera una subred de dos hosts, conectada por un router. Se debe verificar el correcto funcionamiento del fraccionamiento y rearmado de paquetes entre ambos hosts.

Verificar:

$\hat{S}^R$(R1,H1,,,,,1280,,)

pc1 $\rightleftarrows_{B1500}$ pc2

pc2 $\rightleftarrows_{B1500}$ pc1

pc1 $\rightleftarrows_{GB1500}$ pc2

pc2 $\rightleftarrows_{GB1500}$ pc1

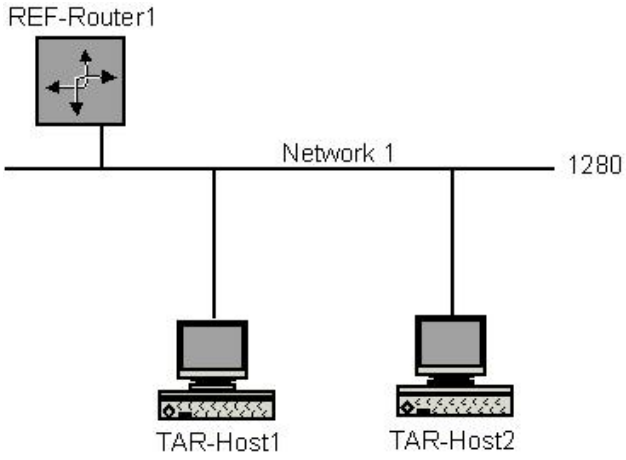


Figura 17: Topología test 1.6.C

Parte D. T1.6.D={H1,R1,R2,R3}, con H1={pc1}, R1 = {r(R2,R2)}, R2 = {r(R1,R3)} y R3 = {r(H1,R2)}. Para este caso se consideran tres subredes conectadas por routers (dos), en un extremo de la red se encuentra un host y en el otro extremo un router. Se debe verificar el correcto funcionamiento del fraccionamiento y rearmado de paquetes entre el host y el router de los extremos de la topología.

Verificar:

$\hat{S}^R$(R1,H1,,,,,,,H3/R2)

$\hat{S}^R$(R2,H2,,,,,,,H3/R3)

$\hat{S}^R$(R3,H2,,,,,,,H1/R2)

$\hat{S}^R$(R3,R2,,(>0),,,,,)

R1 $\rightleftarrows_{GB1500}$ pc1

pc1 $\rightleftarrows_{GB1500}$ R1

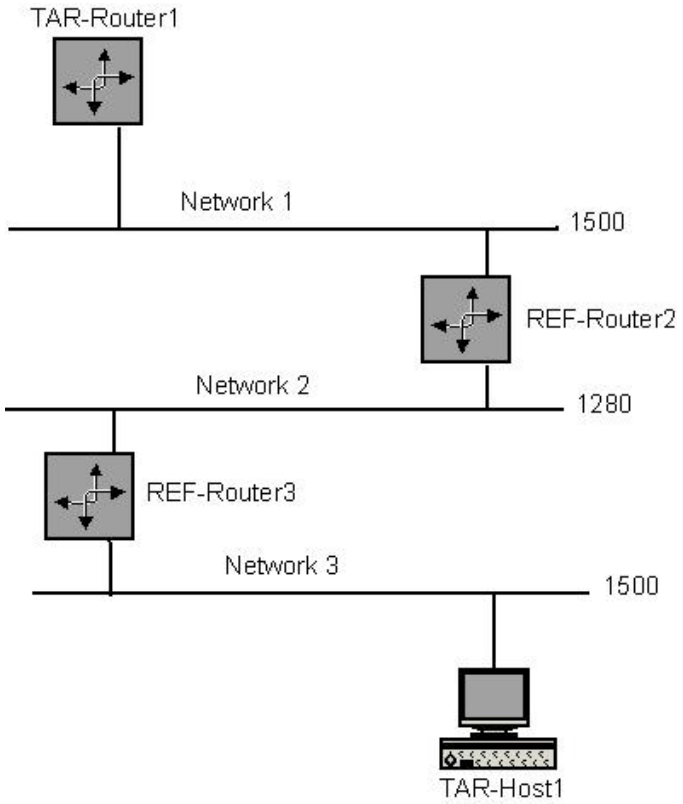


Figura 18: Topología test 1.6.D

Parte E. T1.6.E={R1,R2,R3,R4}, con R1 = {r(R3,R3)}, R3 = {r(R1,R4)}, R4 = {r(R3,R2)} y R2 = {r(R4,R4)}. Para este caso se consideran tres subredes conectadas por routers (dos), en ambos extremo de la red se encuentran routers (2). Se debe verificar el correcto funcionamiento del fraccionamiento y rearmado de paquetes entre los routers de los extremos de la topología.

Verificar:

$\hat{S}^R$(R1,H1,,,,,,,H3/R3)

$\hat{S}^R$(R3,H2,,,,,,,H3/R4)

$\hat{S}^R$(R4,H2,,,,,,,H1/R3)

$\hat{S}^R$(R2,H3,,,,,,,H1/R4)

R1 $\rightleftarrows_{GB1500}$ R2

R2 $\rightleftarrows_{GB1500}$ R1

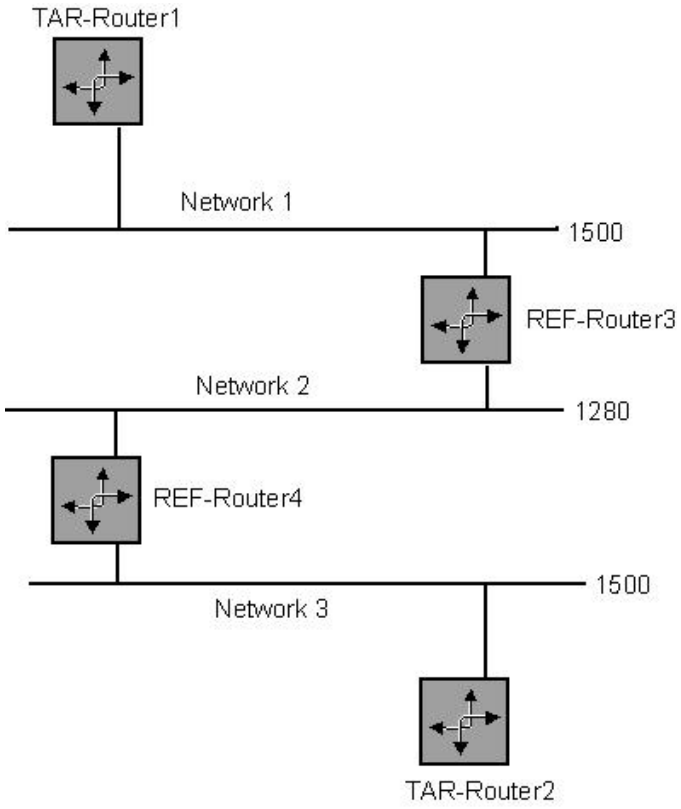


Figura 19: Topología test 1.6.E

4. Implementación

Luego de determinar, y especificar, al menos inicialmente, el modelado con el cual representar los distintos casos de prueba del test de interoperabilidad, pasamos al desarrollo de la posible implementación que refleja dicho modelado.

Para nuestra implementación, y como forma de minimizar en parte los tiempos de implementación, se utilizará como base una implementación ya existente desarrollada por IRISA, realizando modificaciones sobre la misma, con el fin de poder modificar/ampliar la configuración de los casos de prueba.

La re-utilización consiste principalmente en tomar los comandos de sistema existentes en dicha aplicación, es decir, los casos de prueba, son reducidos, por así decirlos, a una serie de comandos a ser ejecutados. Nuestra herramienta, tomará otro camino en lo referente a la lógica para la resolución de la secuencia de comandos, sin embargo, el comando de sistema a ejecutar coincidirá para ambas aplicaciones.

Justificando esta decisión, no parece tener mucho sentido invertir tiempo en determinar como ejecutar a nivel de sistema algunos comandos de configuración de dispositivos, cuando la otra herramienta ya lo hizo. O sea, pese a que es cierto que no podemos afirmar que existen soluciones más elegantes para la ejecución de ciertos comandos, si tenemos la seguridad que los mismos ya fueron testeados y evaluados por otra herramienta.

Actualmente, la herramienta que vamos a modificar para su ejecución, requiere la modificación de tres archivos (en el mejor de los casos de dos de éstos), para poder ejecutar el test adecuadamente. Otro elemento que hace que la herramienta no sea del todo simple de usar, es que para la modificación adecuada de los archivos, se requiere un conocimiento de la herramienta a utilizar, y en particular, para uno de los archivos, conocimientos mínimos, pero conocimientos al fin, de Perl.

Los documentos a modificar para el correcto uso de dicha herramienta son los siguientes:

config_master: donde se indica la configuración de la conectividad para los casos de prueba, ya sea los módulos requeridos para la ejecución, y las distinas vlans y bridges a utilizar.

test.conf: se determinan las distintos host que se van a utilizar para la ejecución del test de interoperabilidad.

plateform.conf: aquí se realiza una breve descripción de los nodos a utilizar para las pruebas, por ejemplo, sistema operativo que corre sobre la máquina, conectividad definida para acceder a la máquina y forma de acceder vía remota al shell (para el caso de utilizar redes virtuales).

Con estas configuraciones definidas, se ejecuta un programa Perl, con el que se genera un directorio donde poder ejecutar el test de interoperabilidad. Finalizada esta instancia, queda en dicho directorio un script de linux, que ejecuta en forma centralizada todas las instrucciones, que permiten, en cierta forma afirmar, luego de analizar los resultados, si el dispositivo a verificar podría pasar los testeos o no.

Es de hacer notar, que la herramienta reporta los resultados de las instrucciones ejecutadas, pero no informa si se producen las salidas esperadas o no.

Es decir para analizar los resultados, como se menciona anteriormente, es necesario además de verificar los resultados de los comandos, realizar un análisis sobre el flujo que ocurre en las redes definidas.

Otro inconveniente que presenta esta herramienta preexistente, es que no es capaz, frente a un cambio en alguno de los tests, de poderlo reflejar mediante algún tipo de configuración, sino que para reflejar esto, la única posibilidad es modificar la programación existente.

Las modificaciones que, en principio, consideramos debíamos realizar a la herramienta para facilitar la configurabilidad eran las siguientes:

- el archivo de configuración debería ser uno solo, y en la medida de lo posible, la interpretación del mismo debería ser lo mas intuitiva posible.
- sería deseable, que la nueva herramienta, no solo muestre las salidas de los comandos invocados, sino que además interprete las mismas, y pueda brindar, al menos aproximadamente, un diagnóstico de cada test.
- en la medida de lo posible y del tiempo que se disponga, la nueva herramienta debería no perder ninguna de las habilidades con que cuenta la ya existente.
- la arquitectura de la solución, no debería diferir mucho de la ya existente, es decir, sería un sistema que podríamos considerar distribuido central, distribuido en el sentido que la ejecución completa se realiza entre varias máquinas (virtuales o no), y central, porque la sucesión de comandos se controlan desde una única máquina.

Una de las cosas que también podemos tener en cuenta, es dotar a la herramienta de la posibilidad de, mediante configuración, no sólo representar los casos de prueba del test, sino poder representar otras pruebas, que creamos necesarias, o que consideremos pertinentes realizar en ese momento.

4.1 Implementación – Puesta en práctica.

Para la implementación, el archivo de configuración que determinará la ejecución de cada test, contendrá la información indicando qué hosts intervendrán en los tests, cuál será la configuración de la red, y cuál será el/los comando/s a ejecutar para la ejecución del test, así como para el caso de las instrucciones que contienen un ICMPv6 Echo Request una expresión regular que determinaría el formato de la expresión a obtener como respuesta que validaría, al menos en una parte, el test.

Inicialmente y como forma de minimizar las tareas para la realización de este proyecto, supondremos que cada uno de los hosts será una máquina virtual, su sistema operativo será un Linux “estándar” (en nuestro caso OpenSuse), entendiéndose por “estándar” un S.O. que acepte los mismos comandos que un OpenSuse para la ejecución de la prueba (ver Apéndice C). Se procederá a realizar una interpretación de las respuestas recibidas, únicamente para los comandos ejecutados. Quedarán para más adelante, y dependiendo del tiempo, interpretar además de las respuestas de los comandos, la interpretación correcta del flujo de red, así como permitir utilizar sistemas operativos que acepten distintos comandos para el manejo de los tests.

Es bueno hacer notar que la solución a implementar permite afirmar si un dispositivo es candidato a pasar el test o no. Utilizamos el término candidato, ya que en realidad, sólo podemos afirmar que en un entorno virtualizable, el dispositivo evaluado, satisface las necesidades requeridas. La imposibilidad de aseverar categóricamente, radica en que los testeos sobre los que se evaluará no es sobre el que se realizan las pruebas requeridas. La práctica, y las pruebas realizadas permiten afirmar que el dispositivo debería satisfacer todos los requerimientos en un entorno real, ya que no existe una razón para descartar la virtualización como una representación del entorno requerido, aunque es justo reconocer que no podríamos afirmarlo en su totalidad hasta probarlo. Es decir, el test que se realiza en los laboratorios para determinar si el dispositivo puede llevar el logo IPv6 Ready, no es posible realizarlo sobre una red virtual, por lo que, pese a que no deberían existir diferencias entre una ejecución y otra, es justo dejar la duda sobre los posibles resultados a obtener.

En base al archivo de entrada que se define para utilizar, se puede incluir dentro de las tareas a realizar, la elaboración de un lenguaje de entrada que fuera posible escribir en un lenguaje casi natural, para luego de una etapa de pre-compilación se generaría el archivo de entrada requerido. Otra opción para analizar, es cambiar el formato de los archivos de entrada, y pasar del formato actual (txt), a un lenguaje más descriptivo (xml).

Dentro de las bibliotecas de testeo, deberían estar las “estándares”, que definen como ejecutar cada cosa, como interpretar cada respuesta, etc. Se debería permitir que el programa pudiera ser “ampliado” con la inclusión de nuevas librerías que cubran necesidades que vayan surgiendo, por ej: ejecución en otros sistemas, que no sean solo virtuales los hosts, etc.

La ejecución se realizará integra desde la maquina anfitriona, es decir la máquina donde se ejecuta el servidor de máquinas virtuales. Dentro del testing, claro está, se realizarán invocaciones remotas utilizando ssh.

Para facilitar la comprensión, una separación básica, utilizando el tiempo como criterio,

podría ser:

- La primera parte del test consiste en ejecutar comandos sobre la máquina anfitriona con el fin de configurar la topología de red requerida para el test a llevar adelante.
- Luego, la ejecución prosigue con la ejecución en las máquinas virtuales, a través de ssh, de los comandos requeridos para cada test. Los retornos de cada invocación remota, son analizados en la máquina anfitriona, como primer medida de correctitud de la ejecución de los tests. Otra medida, utilizada en cada uno de los tests, es un análisis (mediante un parser) del retorno de cada ping6 que se ejecuta.
- La tercera parte consiste en el desarmado (por así decirlo) de la topología de red configurada para el test ejecutado.

Los archivos de configuración que vamos a utilizar serán dos:

* uno que por el momento sería fijo y que indicará que las máquinas a utilizar (cinco en nuestro caso) son virtuales con sistemas tipo Linux (MaquinasDisponibles.pm). En este archivo además se incluye toda la información requerida para detallar la configuración de cada máquina perteneciente al test, si la misma será utilizada como router o como host. En este archivo se hace referencia a otro archivo, de nombre dirs.cfg. Este archivo contiene los nombres de los dispositivos y las direcciones IPv6 por defecto asignadas a cada dispositivo. Esta información existe por separado en un archivo para facilitar la configuración del testeo, ya que se adjunta un script, el cuál se encarga de generar la información de este archivo, para cada máquina donde se desee ejecutar la suite.

* otro que contendría el detalle de las operaciones a realizar para un caso de prueba, algo muy similar a lo indicado en el capítulo **3.3**. Además, en él se puede indicar, mediante una expresión regular, la respuesta esperada para los comandos ICMv6 Echo Request. Dada la simbología definida, para evitar problemas en la representación, es que se realiza una serie de correspondencias de notación, para poder representar algunas operaciones.

Por tanto, y a modo de ejemplificar, la información de configuración correspondiente a un dispositivo está indicada de la siguiente manera:

```
*tn_uy
    eth0=20c:29ff:fec3:a294
    eth1=20c:29ff:fec3:a29e
```

Figura 20: Contenido en dirs.cfg

Las equivalencias de notación a utilizar son:

H1={pc1,pc2} y R1 = {r(H1,H1)}, se puede ver como:

H1={pc1,pc2}

R1={r(H1,H1)}

⇄ equivale a: &&l

$⇄_G$ equivale a: &&g

$⇄_A$ equivale a: &&u

$⇄_{AR}$ equivale a: &&v

Ď() equivale a: &D

$Ŝ^A$() equivale a: &SA

Ě() equivale a: &E

Ŵ equivale a &W

$Ŝ^R$(R1,H1,prefix1,(>0),,,,,) equivale a incluir propiedades en la definición de las propiedades de una conexión del Router. Es decir, para la definición de un router, se utiliza la notación R1={r(H1,H1)}. Si se desea agregar propiedades para una conexión, la notación sería R1={r(H1(<propiedades_de_la_conexión>),H1)}.

Se prevee que en la versión definitiva, se utilizará, para dejar más clara la ejecución de los casos de prueba, una biblioteca de Perl, que será la encargada de proveer direcciones IPv6 con la cual trabajar (Direcciones.pm). Esto será para el caso que no se desee utilizar las direcciones por defecto de cada dispositivo de red, las cuáles, en el caso de ser usadas, como se mencionó previamente, deben ir en MaquinasDisponibles.pm

Las clases, o utilizando terminología Perl, paquetes que se utilizarán son:

Constantes.pm: contiene valores fijos que se utilizarán en distintos momentos del testing. Por ejemplo, los comandos que levantan la conexión tanto de las máquinas destinadas a ser hosts, como las destinadas a ser router.

Direcciones.pm: Contiene, como su nombre lo indica direcciones. Además de las direcciones destino de los ICMPv6 para todos los hosts y todos los routers, contiene un conjunto de direcciones para asignar a las máquinas a utilizar, para el caso que no se deseen utilizar las provistas por defecto por cada máquina virtual y que figuran en MaquinasDisponibles.

MaquinasDisponibles.pm: Ya fue mencionado previamente, en la parte de archivos de configuración, contiene una descripción de las máquinas que serán utilizadas para las pruebas.

General.pm: contiene la lógica para la creación de los comandos a ejecutar para cada línea del archivo que contiene el test a realizar. Aún no está incluido, pero en este archivo es en el que se debería incluir la discriminación según el sistema operativo (y si es virtual o no), los comandos a utilizar.

Inicio.pm: contiene las inicializaciones a realizar para la ejecución de cada test, en este momento sólo se encarga de levantar las conexiones de red mediante la función do_setup.

El programa principal tendrá el nombre: interop.pl, es el encargado de utilizar todas las

otras librerías. Recibe como parámetro un archivo, el cual es un archivo de texto que contiene en cada línea el nombre de otro archivo de texto, que es el que contiene un test a realizar, esta solución fue implementada de esta manera para poder ejecutar mas de un test a la vez. En el caso en que no se ingrese ningún archivo como parámetro, la aplicación supondrá que por defecto el nombre del archivo es testing.txt.

Ejemplos de archivos:

testing.txt:

<Nombres de archivos con cada testeo a correr>//Al menos al comienzo, se aceptará en cada archivo una única configuración de red. Para el caso de cambiar la configuración deberá ingresarse en un archivo nuevo.

Ejemplo de 1 de ellos :

```
pc1:suse1
pc2:tn_uy
R1:suse2
H1={pc1,pc2}
R1={r(H1,H1)}
pc1 &&l pc2 , 5 packets transmitted, [543] received, \d{0,2}% packet loss
pc2 &&l pc1 , 5 packets transmitted, [543] received, \d{0,2}% packet loss{H1,R1}
```

Un flujo de ejecución de un caso de prueba tipo, sería el siguiente:

I. Se inicializan las interfaces de red de cada máquina virtual definida, configurando su uso como host o como router.
II. Se recorre el archivo con los distintos tests a ejecutar. Por cada uno de ellos se ejecuta una función encargada de la ejecución del mismo.
III. Dicha función corrobora el contenido de las líneas a ejecutar, armando una variable con la estructura correspondiente a la topología requerida para el test.
IV. Se procede a la ejecución de los pasos requeridos para cada test. Cada función incluida en la implementación, representa una tarea a realizar. Las funciones son:
    1. borrado de interfaces de red, que pudieran haber quedado de ejecuciones previas (borro_iface).
    2. se agregan las interfaces de red, requeridas para la ejecución del test. El agregado de las interfaces, determina en forma inicial la topología de red requerida para el caso de prueba (levanto_iface).
    3. ejecución del comando tcpdump, el cuál será el encargado de “registrar” el tráfico de red que se realiza sobre cada interface (levanto_tcpdump).

4. asignación de direcciones y levantamiento de las interfaces correspondientes a los nodos definidos como routers para el caso de prueba (levanto_router).
5. se levantan las interfaces correspondientes a los nodos del caso de prueba que fueron definidos como hosts (levanto_host).
6. ejecución de los comandos correspondientes a cada test (ejecuto_comandos). Los posibles comandos a ejecutar son:
   - ping.
   - habilitación de una interface.
   - deshabilitación de una interface.
   - espera (para poder estabilizar la red, luego de algún cambio).
   - asignación de una dirección de red a un dispositivo.
7. dada de baja las direcciones utilizadas para el test (bajo_direcciones).
8. finalización de la ejecución del comando tcpdump (bajo_tcpdump).
9. borrado de las interfaces utilizadas en la ejecución del test (borro_iface).

5. Pruebas

Las pruebas fueron realizadas evaluando la correcta ejecución de todo el sistema de test en una única máquina, es decir las pruebas aplican para una máquina central (la que ejecutará el test), y cinco máquinas virtuales que representan los hosts/routers sobre los que se correrán los tests.

Dichas pruebas fueron realizadas considerando como principal ítem a testear la ejecución correcta de cada uno de los tests de interoperabilidad de IPv6. Es decir, para cada uno de ellos, se verificó que los tests fueran realizados, y que las respuestas a los mismos fueran las esperadas.

Los resultados fueron los esperados, tanto para los de los tests en sí, como para las implementaciones realizadas para las representaciones de cada una de las operaciones existentes en él.

Se presentaron algunos inconvenientes, en lo que respecta a los tiempos de espera para estabilizar la red, luego de la realización de algún cambio que afecta la misma. La realización de los tests, mediante una solución enteramente automatizada, lleva a prestar especial atención a los tiempos de espera para que la red quede correctamente configurada luego de levantar las interfaces de cada dispositivo. La solución consistió en testear bajo distintas cargas en la máquina en que se ejecutó el test, colocando tiempos de espera necesarios para permitir cubrir las situaciones detectadas. En lo que respecta a los tiempos de ejecución de cada comando requerido, se brindó la opción de poder incluir esperas durante la ejecución (mediante el comando &W), con el fin de poder aguardar a estabilizar la red, previo al envío de una solicitud ICMP.

En lo que respecta a la instalación del software requerido para la prueba, se realizó la instalación desde cero en otras máquinas, siguiendo la guía indicada en el Apéndice B, obteniéndose los resultados esperados.

Es necesario consignar, que para el correcto uso de las herramientas, se debe prestar especial atención a las versiones indicadas (Open Suse 11.1 y VMWare Server 2.0.1), ya que se experimentaron problemas, tanto al utilizar otra versión del VMWare, como al actualizar el Open Suse a la versión 11.2.

Dentro del conjunto de pruebas, es interesante resaltar que dentro de las pruebas realizadas, fue posible llevar adelante la suite utilizando Ubuntu 10.04 como sistema operativo de la máquina base. Para este caso, la versión de VMWare siguió siendo la 2.01 y el sistema operativo de las máquinas virtuales era OpenSuse (11.1).

Se realizaron pruebas con el afán de utilizar Virtual Box, como sustituto de VMWare, sin llegar a obtenerse resultados satisfactorios que permitieran el correcto funcionamiento de la suite.

6. Conclusiones.

Se pudo realizar una propuesta de modelado y una implementación completa desde el punto de vista del test a representar para las fases 1 y 2 del mismo.

Se realizó una propuesta de modelado, la cuál permite representar todos los tests pertenecientes a la suite, mediante un conjunto de operadores con sus correspondientes operaciones. La propuesta cubre todas las necesidades de la suite del test de interoperabilidad, así como permite de manera sencilla la ampliación y modificación del modelo a representar.

Brinda un beneficio respecto a la herramienta original provista por IRISA, ya que separa el modelado de la implementación. De esta manera, podemos por un lado definir los modelos de los casos de prueba a representar, para luego transmitir esa información a una implementación capaz de interpretar y por consiguiente ejecutar dichos casos.

En lo que respecta a la implementación, la solución cuenta con dos grandes puntos fuertes. Es capaz de representar no sólo el test, sino que un número bastante mayor de pruebas, cuyo único requisito para su funcionamiento, es que utilicen las mismas funciones descritas en el test. Además, es capaz de reflejar de manera rápida cualquier cambio en la descripción del test, ya que una modificación implicaría cambiar archivos de configuración y no código ejecutable propiamente dicho.

Tiene la ventaja que es capaz, al menos parcialmente, de interpretar los resultados obtenidos de los comandos ejecutados. En lo que respecta a este ítem, queda pendiente para trabajos a futuro poder automatizar la interpretación del flujo de datos que pasa por la red.

Otro ítem que es posible mencionar, es que el mismo está realizado en su totalidad en Perl, sin la utilización de ninguna biblioteca especial dependiente del sistema operativo, es decir, la gestión del test es posible ejecutarlo en cualquier sistema operativo en el que se pueda correr Perl, lo cuál da un espectro bastante amplio de potenciales máquinas a utilizar. Es necesario mencionar que dentro del código se realizan invocaciones a comandos del sistema operativo, por lo que podríamos afirmar que es mayor la limitante en este sentido, que en el del uso de Perl. Resolviendo las equivalencias de los comandos de sistemas entre un sistema operativo y otro, la portabilidad de Perl en distintos sistemas operativos es un detalle a tener en cuenta.

Es bueno resaltar, que la operativa, no sólo permite validar los tests del IPv6 Ready Logo. Mediante esta solución, es posible testear una red virtual cualquiera. De hecho, si montáramos una red física (no virtual), realizada con dispositivos virtuales, o si concluimos dentro de los trabajos pendientes la posibilidad de poder realizar los tests sobre una red física, la solución nos permitiría validar la inter-conexión de una red conformada hasta por tres sub-redes. La suite permitiría, mediante un archivo de configuración determinar problemas, o por defecto validar, la conectividad entre varias máquinas utilizando IPv6 como forma de comunicación.

Dentro de las herramientas adicionales realizadas, con el fin de facilitar su uso, se incluyó un script inicial que realiza una especie de pre-proceso, a ejecutar con anterioridad a la ejecución de los tests en sí. La finalidad del mismo, es brindar la información que puede llegar a quedar pendiente, para el armado de la configuración necesaria para la ejecución de los tests. Dicho script, permite obtener las direcciones

que tiene por defecto asignado cada dispositivo interviniente en el test, para de esta manera, independiente de las MAC Address que asigne la máquina virtual, la herramienta es capaz de ejecutarse sin tener que modificar ninguna librería.

7. Trabajo Futuro.

A continuación, es bueno mencionar qué tareas quedan pendientes para potenciales trabajos a futuro. Dichas tareas, dotarían a esta herramienta de un mayor poder de configuración, y completitud, en lo que respecta a sus potenciales habilidades.

La totalidad de las tareas postergadas para trabajos a futuro, fueron producto más que nada de la imposibilidad de cumplir con las mismas en la duración de un proyecto de grado.

No se pudo detectar que existan otros impedimentos, además del tiempo, para llevar adelante las tareas pendientes. Como forma de facilitar el detalle, se decide clasificar las tareas pendientes en distintas áreas temáticas.

Podemos considerar como pendiente, analizar los cambios a aplicar, para poder lograr que el test pueda correr sobre una red física, y no virtual como la utilizada en este proyecto. Por los pasos realizados en la implementación, todo parece indicar que, cambiando mediante configuración la primera y última parte de los test, donde se levantan los dispositivos virtuales, la aplicación debería poder correr sin problemas. Para poder trabajar sobre esta parte, se torna necesario, contar con una red física que pudiera cubrir todas las necesidades de los distintos escenarios de prueba.

Otro ítem a mencionar que queda pendiente, y sin duda, que es de los más interesantes sobre los cuales proseguir, es permitir que el test, pueda correr sobre otros sistemas operativos. Básicamente, el concepto de “otros sistemas operativos”, aplica a los sistemas operativos que utilizan comandos de ejecución de las distintas partes del test, distintos a los manejados por la versión de Linux (OpenSuSE) utilizada. La solución debería ir por el lado de definir para cada nodo participante del test, el sistema operativo con el que cuenta, para que en base a el mismo, se pudiera contar con el juego de comandos necesarios para la realización del test adecuadamente.

Como se menciona en el capítulo de pruebas, puede ser interesante evaluar la posibilidad de ejecutar la suite sobre otros sistemas operativos, y sobre otras herramientas de virtualización, con el fin de ampliar el espectro de ejecución de la solución propuesta.

Referencias bibliográficas

1) IPv6 READY. Phase 1/2 Test Interoperability Specification Core Protocols. Technical Document. Revision 4.0.1. - PDF – (2008)

2) IPv6 Ready Logo. Phase-1/2 Interoperability Test Scenario Core Protocols. Appendix A. Technical Document. Revision 4.0.1. - PDF – (2009)

3) A Framework for Interoperability Testing of Network Protocols / Alilovic-Curgus, Vuong (1993) - PDF - Technical report, Dept. of Comp. Sci., University of British Columbia, *1993* .

4) A Test Specification Method for Software Interoperability Test in OffShore Scenarios: A Case Study / Sakout Andaloussi, Braun (2006)- IEEE Computer Society - Proceedings of the IEEE international conference on Global Software Engineering - Florianópolis, Brasil – 16 al 19 de octubre de 2006 - páginas 168-178.

5) CBR Support for CSP Modeling of InterOperability Testing / Sqalli, Freuder. - PDF - Workshop on CBR Integrations, AAAI, Madison, Wisconsin, USA (1998).

6) Towards Interoperability Test Generation of Time Dependent Protocols: a Case Study / Zhiliang, Jianping, Xia (2004). Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE – páginas 589-594 (volumen 2). 29 de noviembre al 3 de diciembre de 2004.

7) IPv6 Conformance and Interoperability Testing (2005) - IEEE Computer Society - Joan Ruiz, Àlex Vallejo, Jaume Abella , 10th IEEE Symposium on Computers and Communications (ISCC'05), 2005 pages 83-88. Cartagena, Murcia, España – 27 al 30 de Junio de 2005.

8) State of the Art of IPv6 Conformance and Interoperability Testing (2007). Vallejo, A.; Ruiz, J.; Abella, J.; Zaballos, A.; Selga, J.M.; Communications Magazine, IEEE – Octubre 2007 - páginas 140-146.

9) "Virtualized Interoperability Testing: Application to IPv6 Network Mobility" Springer - A. Sabiguero, A. Baire, A. Boutet and C. Viho - DSOM 2007: 18th IFIP International Workshop on Distributed Systems: Operations and Management. Managing Virtualization of Networks and Services (held as part of

Manweek 2007) - 29-31/10, San José, California, United States of America, 2007, ISBN 978-3-540-75693-4, pages 187-190. Springer 2007.

10) "Towards an IP-oriented testing framework - The IPv6 Testing Toolkit"- PDF -A. Sabiguero, A. Baire, A. Desmoulin, A. Floch, F. Roudaut and C. Viho - TTCN-3 User Conference 2006 - 30/5-2/6, Berlin, Germany, 2006.

11) "Plug once, test everything. Configuration management in IPv6 Interop Testing." IEEE Computer Society - A. Sabiguero and C. Viho - Proceedings of the fifteenth Asian Test Symposium - Hiromi Hirashi, Hideo Tamamoti, Seiji Kejihara (Editors) - 20-23 Nov., Fukuoka, Japan, 2006, ISBN 0-7695-2628-4, pages 443-448, IEEE Computer Society Order Number P2628.

12) RFC 791. Internet Protocol. Setiembre 1981. Web

13) RFC 1883. Internet Protocol, Version 6 (IPv6). Specification. Diciembre 1995. Web

Apéndices

A) Opciones pre-seleccionadas.

Brevemente, se procede a describir las opciones pre-seleccionadas.

Intop

Basado [3].

Se trata de un documento que data de 1993, en su contenido se procede a describir un modelado, al cual se le da el nombre intop, como una posible opción para poder realizar test de interoperabilidad para protocolos de red.

Esta teoría es posible de aplicar, para, a partir de métodos formales de especificaciones, poder representar la solución en términos de un sistema de transición etiquetado.

Intentando una calificación para el procedimiento, podemos considerar que este método se encuentra dentro de los que podemos llamar procesos algebraicos.

Para la definición del modelado, utiliza conceptos similares a los de un test de conformidad.

Requiere que, para la especificación del protocolo, la definición del test, esté dada por un proceso algebraico, al estilo de LOTOS, es decir un sistema de transiciones etiquetadas de entrada/salida.

Realizando una lectura del contenido de este paper, es posible determinar que la utilización de sistemas de este tipo, no son de las más apropiadas para la puesta en práctica y representación de un test de interoperabilidad y sus potenciales variantes mediante este método.

Por los ejemplos incluidos en este paper, se puede presuponer que, muy probablemente, la representación de algunos de los escenarios del test de interoperabilidad serían demasiado extensos en su representación para su posterior ejecución.

U2TP

Basado en referencia [4].

Siguiendo con el análisis de los posibles casos de modelado a seleccionar, pasamos a un enfoque bastante diferente con respecto al anterior. Para este caso, el modelado sugerido para distintos casos de prueba es utilizando UML-2 Testing Profile (U2TP) en conjunción con TTCN-3.

U2TP esta basado en conceptos UML, y es una suite que contiene una serie de diagramas Uml los cuales se utilizan para poder representar los casos de prueba, mediante casos de uso, diagramas de sistema, etc.

De esta manera, es que podríamos utilizar UML para modelar todos los casos de prueba indicados en el test de interoperabilidad.

Dentro del documento se especifica que, luego de representar el caso de prueba mediante Uml, la opción sugerida para poder, digamos, mapear la implementación y ejecución de los mismos es mediante TTCN-3.

En resumen, en este documento, se indica como una buena práctica, la opción de poder hacer comprensibles los modelados mediante el uso de conceptos abstractos, representados por diagramas UML. El inconveniente, en este caso, surge por la dificultad de transformar, por así decirlo, el diagrama generado en UML a una implementación.

En el documento, no se detalla la forma de poder realizar mediante un “método directo” (por ej. un algoritmo) el pasaje de UML a una implementación. Es decir, mediante Uml la comprensión del caso de prueba se vuelve bastante simple, pero la transición de la “idea” (diagrama) a una implementación, no parece ser muy sencilla de realizar de una forma automatizada, o al menos semi-automatizada.

CSP/CBR

Basado en [5].

Prosiguiendo con la lectura de los papers, en este caso, la sugerencia para el diagnóstico de un test de interoperabilidad, consiste en integrar un razonamiento basado en condiciones y sus casos.

En así, que se indica como una posible solución la elaboración del test de interoperabilidad utilizando la combinación de CSP (Constraint Satisfaction Problem) con CBR (Case-Based Reasoning).

La operativa consiste en, para resolver el problema, utilizar CSP, y para el caso en que el modelo esté incompleto o incorrecto, el mismo se resolvería con la utilización de CBR.

Los modelados que podríamos realizar, podrían no estar completos, ya que representan el funcionamiento de un sistema y por tanto pueden no estar incluidas la totalidad de las interacciones del sistema.

Además, los modelos pueden tener incorrecciones producto que los mismos representan sistemas que contienen defectos (errores o bugs) o simplemente no están bien definidos.

CSP es utilizado en muchas situaciones, ya sea para diagnosticar problemas, como para representar interoperabilidad de sistemas, y en resumen es un grafo en el cual los vértices son variables del problema y las aristas son condiciones entre las variables.

Si la representación obtenida del sistema con CSP, no fuera suficiente, podríamos complementar la misma, utilizando CBR, herramienta que usualmente es utilizada cuando no hay suficiente información empírica.

La solución vendría por el lado de inicialmente modelar la especificación brindada, mediante CSP para ver si es posible resolver el problema utilizando únicamente esta herramienta.

En el caso que el modelo halla quedado incompleto, utilizamos CBR, como forma de poder actualizar y/o corregir el modelo.

CBR cuenta de los siguientes pasos principales:

1. primero identificar si ya existe en nuestro “conocimiento” un problema similar al que hay que resolver.
2. si es así, recuperamos la solución del mismo,
3. la adaptamos a nuestras necesidades, y
4. almacenamos esta solución para futuros problemas similares.

CTIOA

Basado en [6].

Para este caso, la opción sugerida consiste en la aplicación de los siguientes pasos.

Como primera parte, se debería calcular, es decir obtener, un autómata el cual representa el caso a testear, utilizando un método de análisis de alcance sobre el cuál se aplicará el caso de prueba, para luego analizar las distintas secuencias de testeo.

Al obtener las distintas secuencias, se pueden descartar las que en la práctica no ocurran, o las que estuvieran representadas, por así decirlo, en otras secuencias, para de esta manera identificar y quedarnos con las potencialmente ejecutables.

De esta manera podemos reducir nuestro problema a uno lineal, es decir, podemos ejecutar el test en forma secuencial, seleccionando los valores de inicio de forma adecuada.

Uno de las principales técnicas para el modelado y representación de sistemas que utilizan el tiempo como una variable a ser tomada en cuenta, se refiere a los mecanismos identificados como Autómatas de Tiempo.

Podemos definir un CTIOA, para nuestro modelado como un conjunto de TIOA's (es decir Autómatas de Entrada Salida con temporizadores) y un conjunto de canales “C”, que son los encargados de comunicar los distintos TIOA's entre sí.

Cada uno de las TIOA esta formado por un conjunto de estados, un conjunto de entradas y salidas, un estado inicial, un conjunto de clocks y un conjunto de transiciones entre estados.

Dicha idea consiste en definir grafos que representan distintos nodos, para luego poderlos componer, con el fin de definir un grafo de alcance global.

Luego de obtenido el mismo, la tarea prosigue con analizar si el mismo tiene caminos que puedan ser ejecutados, es decir que las restricciones establecidas de tiempo puedan ser cumplidas.

Para cada comunicación de a pares (entre 2 TIOA, comunicados por un canal), está claro que lo que corresponde a una entrada en uno de ellos, debería corresponder a una salida en el otro.

De esta manera podemos “combinar” ambas transiciones, sincronizándolas, es decir, ir armando por cada par un CTIOA, para terminar en un gran CTIOA, el cual contendría todas las comunicaciones e interacciones correspondientes al caso a analizar.

Con esta situación, es que ahora debemos ir identificando las posibles ejecuciones lineales de los posibles casos de prueba, es decir que casos de los indicados previamente pueden ser sincronizados, para de esta manera poder realizar la ejecución de los mismos.

En este caso, la solución final, sería la ejecución lineal de los distintos casos identificados.

B) Manual de Usuario

A continuación se detallan los pasos significativos a seguir, para poder replicar las pruebas realizadas en condiciones similares a las que fueron elaboradas y realizadas.

Pre-condiciones.

En lo que se refiere a hardware, la mayor parte del tiempo se utilizó un Intel Celeron 2.26Ghz, con 1 Gb de memoria y un disco Sata de G0 Gb de capacidad, por lo que en base a los equipos disponibles en el mercado, considero que el hardware no debería ser una limitante para la puesta en práctica de los tests. La mayor restricción existente es la cantidad de memoria que consumen las máquinas virtuales, principalmente en el booteo de las mismas, por lo que, contando con una máquina con la capacidad suficiente para bootear cinco máquinas virtuales, los requerimientos estarían cubiertos. El consumo de cualquiera de estas máquinas virtuales es elevado al iniciarse, pero luego es muy bajo, además, el consumo de recursos de los tests realizados no es significativo. Lógicamente, con una máquina de mayores prestaciones, el inicio de las máquinas virtuales es más rápido, producto de asignarle más memoria a cada una de ellas, disminuyendo los tiempos de espera para poder ejecutar los tests.

En cuanto a software, para la ejecución de pruebas, se utilizó como Sistema Operativo Open Suse 11.1 y como herramienta de virtualización VmWare Server v2.0.1. Para la ejecución de nuestros tests, requeríamos hacer funcionar correctamente nuestros bridges virtuales en VmWare Server, es por esta razón que se debe aplicar un patch, elaborado con anterioridad por el tutor de este proyecto, incluido en el sub-directorio parche_vmserver, siguiendo las indicaciones del archivo “como.txt”.

Se procede a instalar cinco máquinas virtuales, con igual sistema operativo que el base (para las tareas a realizar podría ser cualquier sistema Linux similar a OpenSuse). Las prestaciones de las máquinas creadas fueron las siguientes: 192 Mb de memoria, 8 Gb de disco y 3 tarjetas de red.

Para la utilización del armado de redes, se define el siguiente criterio:

- Cada máquina virtual tendrá tres conexiones de red, dos para las posibles conexiones de los test de interoperabilidad (conexiones IPv6), y una para la recepción de los comandos a ejecutar (IPv4), que será compartida por todas las máquinas virtuales.
- Todas las redes virtuales a utilizar se implementarán sobre la red eth0. Vale la pena recordar, que para el caso de sistemas Linux, levantar los servicios de eth0 implica tener conexión de red.
- Para ser utilizada como enlace de red para la transmisión de comandos (IPv4) utilizaremos eth0.30.
- Luego utilizaremos la secuencia de redes desde eth0.10 hasta eth0.19, donde eth0.10 y eth0.11 (además de la ya definida eth0.30) corresponden a la máquina virtual 1 y así sucesivamente.
- De esta manera cada maquina virtual tendrá definidas tres redes, eth0, eth1 y eth2. Para todas las máquinas virtuales eth2, corresponderá a la definida como eth0.30. Luego, y a modo de ejemplificar para la máquina virtual 1, eth0

corresponderá a eth0.10 y eth1 a eth0.11.

- De los casos de prueba analizados, el máximo de sub-redes sobre las cuales se deben ejecutar tests es de tres, razón por la cual definimos para el desarrollo de las pruebas, tres bridges con los cuáles trabajar.
- En lo referente a las invocaciones, ejecutaremos los comandos ssh refiriéndonos a las máquinas por su nombre de host y no por su dirección. Para poder realizar esta tarea deberemos editar el archivo /etc/hosts e incluir una línea por cada máquina virtual con la relación entre hostname y dirección. A modo de ejemplo:

  192.168.66.100 OpenSuse

- Para la ejecución de los comandos desde una máquina a otra, utilizamos ssh para el envío de la instrucción. Cada máquina virtual tiene definido al usuario root, habilitado para enviar instrucciones desde otra máquina, utilizando como método de autenticación clave pública-privada. De esta manera evitamos tener que indicar la password al enviar cada comando desde nuestra máquina a las máquinas virtuales. Este paso se puede ubicar en detalle en un sinfín de páginas de Internet, por lo que en este lugar haremos una breve indicación, de una de las posibles maneras de realizar esta tarea: desde la máquina que mandaremos a ejecutar todos los comandos, creamos la llave utilizando: 'ssh-keygen -t dsa -b 1024', de esta manera obtendremos la clave privada (id_dsa) y la pública id_dsa.pub. A continuación deberemos copiar la clave pública (id_dsa.pub) a la máquina que realmente ejecutará los comandos. Para que todo funcione correctamente, deberemos copiar el archivo en el sub-directorio '.ssh' del usuario con el nombre authorized_keys. Por ejemplo invocando: .

  scp id_dsa.pub root@OpenSuse:.ssh/authorized_keys

  Dada la necesidad de ejecutar algunas instrucciones que requieren de permisos privilegiados, es que la ejecución de los tests, la realizamos con el usuario root. No debería haber problemas si se desea utilizar otro usuario, pero se debe tener en cuenta si el mismo tiene los permisos suficientes. Algunas de las instrucciones que se ejecutan son: tcpdump, brctl. Por mas información ver Apéndice C.

En nuestro caso, y previo a la inicialización de las máquinas virtuales, con el fin de poder levantar todas las interfaces de redes a ser potencialmente usadas, es que ejecutamos la siguiente secuencia de comandos (esta secuencia se encuentra incluida dentro del archivo eths_up):

ifconfig eth0 up #Levanto eth0 por las dudas que no este ya levantado

modprobe 8021q

vconfig add eth0 10

vconfig add eth0 11

vconfig add eth0 12

vconfig add eth0 13

vconfig add eth0 14

vconfig add eth0 15

vconfig add eth0 16

vconfig add eth0 17

vconfig add eth0 18

vconfig add eth0 19

vconfig add eth0 30

ifconfig eth0 0.0.0.0 up #Quito la Ip que tenga asociado eth0

ifconfig eth0.30 192.168.66.100 up #Le asigno una IP a eth0.30, voy a pasar comandos por IPv4.

ifconfig eth0.10 up

ifconfig eth0.11 up

ifconfig eth0.12 up

ifconfig eth0.13 up

ifconfig eth0.14 up

ifconfig eth0.15 up

ifconfig eth0.16 up

ifconfig eth0.17 up

fconfig eth0.18 up

ifconfig eth0.19 up

ifconfig eth0.30 up

brctl addbr Network1

brctl setfd Network1 0

brctl addbr Network2

brctl setfd Network2 0

brctl addbr Network3

brctl setfd Network3 0

ifconfig Network1 0.0.0.0 up

ifconfig Network2 0.0.0.0 up

ifconfig Network3 0.0.0.0 up

Luego de los pasos mencionados, procedemos a inicializar las máquinas virtuales.

En nuestro caso, definimos las 5 máquinas virtuales con los siguientes nombres de host.

suse1: Ip=192.168.66.10, máquina definida en los tests a ser utilizada como host.

suse2: Ip=192.168.66.20, máquina definida en los tests a ser utilizada como host.

suse3: Ip=192.168.66.30, máquina definida en los tests a ser utilizada como router.

suse4: Ip=192.168.66.40, máquina definida en los tests a ser utilizada como router.

tn_uy: Ip=192.168.66.50, máquina definida en los tests a ser utilizada como host.

La configuración requerida para el test, implica el conocimiento de los nombres de los hosts de cada máquina virtual, y las direcciones IPv6 que se asignan al levantar cada interface. Además, en el directorio Archivos se encuentra un archivo por cada uno de los tests de interoperabilidad, donde aparece en cada uno de ellos el nombre de las máquinas virtuales que intervienen en los mismos.

Para facilitar estas tareas, se provee el script de nombre config ubicado en la raíz del proyecto. Dicho script debe ser ejecutado con las máquinas virtuales levantadas, y con conexión de red.

El mismo, requiere de cinco parámetros para su invocación, los dos primeros, dos máquinas que por defecto serán utilizadas como hosts, los dos siguientes, máquinas que por defecto serán utilizadas como routers, y el quinto y último será utilizado como host, y será el equivalente de la máquina a testear (implementation under test).

Dicho script, creará el archivo dirs.cfg, que contiene las direcciones IPv6 de eth0 y eth1 para cada máquina virtual, utilizando a tales efectos el Perl armo_eths.pl. A continuación, el mismo script ejecutara el script del directorio Archivos de nombre cambiar_nombres_hosts. Dicho script configura cada archivo de test para poder ser ejecutado con los dispositivos provistos.

Para este caso el mismo fue ejecutado de la siguiente manera:

```
sudo ./config suse1 suse2 suse3 suse4 tn_uy
```

Operativa

Luego de realizadas todas las tareas indicadas como pre-condiciones estamos aptos para la realización de los test de interoperabilidad.

El programa principal está en el archivo interop.pl, por lo que la ejecución de los test, se realizan invocando el comando:

```
./interop.pl <archivo> <cambio_de_uso>
```

En lo que se refiere al parámetro archivo, el programa acepta un archivo con la lista de los distintos test a realizar. En el caso en que no se ingrese ningún nombre, o el mismo sea 0 (cero), el programa asume por defecto que el nombre del archivo es ***testing.txt***.

El parámetro cambio_de_uso, es una posibilidad que existe para permitir que alguna de las máquinas intervinientes en el test, pueda cambiar el uso que tiene asignado por defecto. A modo de ejemplo, como mencionamos previamente, el host definido como suse2, lo íbamos a utilizar como host, si por alguna necesidad lo quisiéramos usar como router, para un test específico, bastaría con agregar como parámetro para la invocación de interop.pl, el siguiente parámetro: ***suse2=router***. La utilización de este parámetro extra se utiliza para los tests 1.6.D y 1.6.E, la ejecución de dichos tests se realiza invocando los scripts: ejec_1.6.D y ejec_1.6.E respectivamente. Como aclaración ambos

scripts, sobrescriben el contenido del archivo testing.txt.

En lo que se refiere al archivo ingresado como parámetro, el mismo contiene una lista de los archivos con los distintos test que se desean correr para esa ejecución.

Un contenido posible, es el siguiente:

```
#----------------------testing.txt--------------------
#Otro comentario
   #Comentario
#Lo que sigue es una linea vacia

#Lo que sigue es una linea que contiene solo espacios

../Archivos/1.1.A
../Archivos/1.1.B
#----------------------testing.txt--------------------
```

En este caso este archivo ejecuta el contenido de los archivos de nombre '1.1.A' y '1.1.B', que se encuentran en el directorio Archivos del proyecto.

El contenido de cada archivo de test es una de las siguientes líneas indicadas a continuación, recordando que cada línea que comienza con '#' es un comentario.

Las primeras líneas indicarán las máquinas a usar y a que máquina del test están asociadas, por ej:

```
pc1:suse1
pc2:tn_uy
R1:suse2
```

Luego se definirán la estructura de red a utilizar para el test, por ej:

```
H1={pc1,pc2}
R1={r(H1,H1)}
```

indicará que se trata de un router (R1) y una única sub-red compuesta por dos máquinas.

Otra forma mas compleja de utilizar la línea R1, es la siguiente:

```
R1={r(H1(*),H2(*))}
```

donde se indica que para la conexión del router, se le pueden indicar algunas propiedades a la misma, cada una de ellas tendrá el valor: propiedad{valor}.

Luego siguen los comandos a ejecutar con la realidad definida previamente.

Los Echo Request (ping) se representan mediante el siguiente comando '&&', y su

formato es el siguiente, por ej:

```
pc1 &&g pc2 , 5 packets transmitted, 0 received, 100% packet loss
```

la primera parte indica las dos máquinas que van a ejecutar el comando (origen $\rightleftarrows_G$ destino). Los posibles comandos son g,l,u, donde g indica que su usa la direccion global, l la local, y u la dirección multicast. Lo que sigue es un texto (expresión regular) de la respuesta que se espera recibir al ping a realizar.

Los comandos utilizados, y que no son del tipo Echo Request, son los siguientes:

&W(X,Y), indica que se debe esperar Y segundos para proseguir a la siguiente línea del test. El código también permite que si se ingresa X con el valor Enter, el test se quedará esperando hasta que el operador presione Enter.

&D(A,1), indica que se debe deshabilitar la máquina A, la conexión correspondiente a la sub-red 1. En el caso de más de un par, la ejecución se realiza de izquierda a derecha.

&E(A,1), igual al anterior, sólo que en este caso se habilita la conexión.

&SA(A,B), se le asigna al host A, la dirección del host B.

&W(Enter,) o &W(,N), indica que se debe esperar, en el primer caso hasta presionar la tecla enter, en el segundo caso N segundos.

Herramientas opcionales

Una de las tareas pendientes de realización, para dotar de herramientas de diagnóstico más refinadas, consistía en hacer un análisis de las salidas obtenidas, mediante la utilización de tshark. Dado que no se pudo disponer del tiempo necesario, para incluir esta funcionalidad, se agrega un script, y un pequeño programa Perl, que contiene una primera aproximación. El script se encuentra ubicado en la raíz del proyecto, y se llama ejec_tshark, debiéndose pasar como parámetro el número de subred a analizar. Este comando genera un archivo de salida de nombre salida_tshark_n<SUBRED>.txt, el cuál contiene todos los paquetes que circularon durante esa subred, durante el tiempo en que estuvo ejecutándose el programa. El programa Perl de nombre filtrar_tshark.pl, que también se encuentra en la raíz del proyecto, es la otra parte a utilizar para realizar el análisis básico. Dicho programa, en base a un archivo de salida del script anterior, que es pasado como parámetro, indica en pantalla los mensajes Echo request y Echo reply que contiene el mismo. De esta manera, es posible analizar, en forma básica, el tráfico que se produce en cada subred.

Un posible uso que se le puede dar, al menos como esta elaborado actualmente, es el siguiente: antes y después del comando ping6 que se desea analizar, colocaríamos la instrucción &W(Enter,), es decir esperar hasta pulsar Enter. Cuando en el flujo del test se llega a la espera previa a la ejecución del comando, deberíamos ejecutar el script ejec_tshark con la subred a a analizar (o tantas ejecuciones como subredes deseáramos chequear), luego presionar Enter, esperar la ejecución del comando, y luego, cuando apareciera la espera por Enter, posterior al comando, detener la ejecución de los comandos tshark ejecutados previamente. De esta manera tendríamos uno, o varios, archivos de texto con las salidas de cada subred, para analizar, ya sea revisando su contenido, o ejecutando el programa Perl proporcionado, como una posible primera aproximación.

C) Anexos

Comandos que debe aceptar el sistema Linux para la correcta ejecución del test.

A continuación, se listan los comandos que el sistema operativo sobre el que se van a a correr los tests, debe ser capaz de ejecutar:

tcpdump: herramienta para analizar tráfico en la red (comando existente en prácticamente cualquier UNIX: Linux, Solaris, BSD, AIX).

brctl: para el manejo de bridges.

ifconfig: para la configuración de las interfaces de red.

route: para el enrutamiento de un sistema.

sysctl: para leer y escribir parámetros del sistema.

radvd: router advertisement daemon for IPv6.

D) Glosario:

IEEE: The Institute of Electrical and Electronic Engineers
LOTOS: IOLTS: Inputs/Outputs Labeled Transition Systems.
UML: Unified Modeling Language.
TTCN-3: Testing and Test Control Notation, versión 3.
CSP: Constraint Satisfaction Problem.
CBR: Case-Based Reasoning.
IETF: Internet Engineering Task Force (IETF) (en español Grupo de Trabajo en Ingeniería de Internet).
NREN: National Research and Education Network.
Perl: Practical Extraccion Report Language.