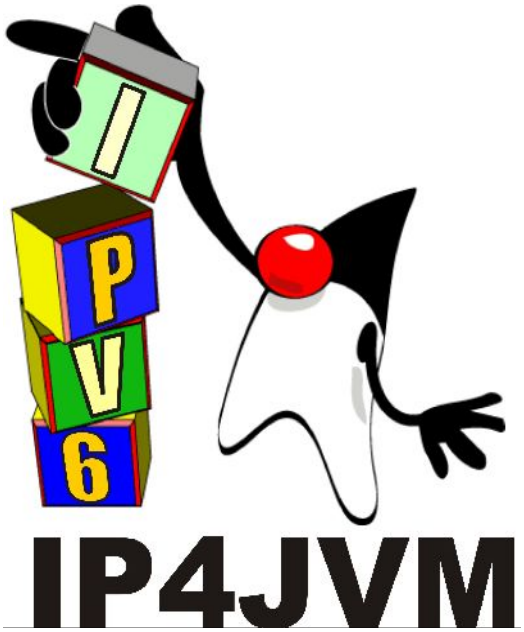

Proyecto de Grado 2010

CERTIFICACION DE IP4JVM

Anexo IV –Tests de conformidad

Autor: Daniel Enrique Rosano Lorenzo

Tutores:
Ariel Sabiguero Yawelak
Leandro Scasso

Instituto de Computación

Facultad de Ingeniería

Universidad de la República

Octubre de 2011

1 Índice

Introducción

En este anexo se discuten las soluciones a los tests de conformidad que fallaron inicialmente, describiendo:

- Propósito del test: se describe el caso de prueba que va a cubrir dicho test.
- Procedimiento del test: se detallan los pasos que realizan las máquinas de prueba (TN) y la máquina bajo test (NUT).
- Resultados esperados (Criterio PASS/FAIL): se indican los resultados que se deben obtener para que el test resulte exitoso.
- Resultado obtenido: ya que solo se muestran los resultados de los tests fallidos se detalla la posible razón del error y si es posible se muestra el paquete recibido (en algunos casos el error consiste en la no recepción de paquetes).
- Solución: se detalla la solución al error, ya sea explicando el código modificado o refiriendo la solución a algún otro test con soluciones similares.

2 Section I - IPv6 Specification

Los siguientes tests cubren la especificación básica del Internet Protocol version 6, como se describe en el RFC (Request For Comments) 2460.

La especificación básica describe el IPv6 header y los Extension headers y options. Discute los problemas con los tamaños de paquetes, la semántica de las Flow Label y del Traffic Class, y por último los efectos de IPv6 en protocolos de capas superiores.

En esta sección fallaron inicialmente 15 tests.

1	-	15	PASS	29	PASS	43	PASS
2	PASS	16	FAIL	30	PASS	44	FAIL
3	PASS	17	FAIL	31	PASS	45	FAIL
4	PASS	18	FAIL	32	PASS	46	FAIL
5	PASS	19	PASS	33	PASS	47	PASS
6	PASS	20	PASS	34	PASS	48	FAIL
7	PASS	21	PASS	35	PASS	49	FAIL
8	FAIL	22	PASS	36	PASS	50	FAIL
9	PASS	23	PASS	37	PASS	51	FAIL
10	PASS	24	PASS	38	PASS	52	PASS
11	FAIL	25	PASS	39	FAIL	53	FAIL
12	PASS	26	PASS	40	PASS	54	PASS
13	PASS	27	PASS	41	FAIL		
14	PASS	28	PASS	42	PASS		

2.1 Test 1.8 - Unrecognized Next Header in IPv6 Header (Multiple Values)

2.1.1 Propósito

Verificar que un nodo, genera una respuesta apropiada en caso de recibir un paquete con el campo Next Header con un valor desconocido.

2.1.2 Procedimiento

1. TN transmite un Echo Request que tiene un IPv6 header con el campo Next Header en 138 (desconocido).
2. TN transmite un Echo Request válido solicitud al NUT.
4. Repetir los pasos 1 a 3 con todos los valores no reconocidos para Next Header entre 139 y 252 en el paso 1.

2.1.3 Resultados esperados

3. El NUT debe enviar un mensaje ICMPv6 Parameter Problem a TN1 tal que
- El campo Code = 1(unrecognized Next Header type).
- El campo Pointer = 0x06(offset del campo Next Header).

2.1.4 Resultado obtenido

Paquete recibido:

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

Razón de la falla: *"The ICMPv6 Pointer field should be 0x06(offset of the Next Header field)."* En este caso el test falla ya que el valor del campo Pointer dentro del paquete ICMPv6 es (0xFFFFFFF8) 4294967288 no es 6.

2.1.5 Solución

Modificar la clase IPv6Protocol línea 162.

```
sendParameterProblem(ns.getNetconfig(),
ICMPv6PProblem.UNRECOGNIZE_NEXTHEADER,netMsg.getBytesfromOffset() ,
nextHeadOffset-netMsg.getOffset());
```

```
nextHeadOffset (6) - netMsg.getOffset() (14) vale -8 (valor incorrecto
para el pointer ya que debería valer 6)
```

Para que el test pase se debe modificar la línea 162 que asigna `el valor de nextHeadOffset`

```
nextHeadOffset=
IPv6ProtocolHeader.VER_TRAFF_FL_WIDTH+IPv6ProtocolHeader.LENGTH_WIDTH;
```

a

```
nextHeadOffset=
IPv6ProtocolHeader.VER_TRAFF_FL_WIDTH+IPv6ProtocolHeader.LENGTH_WIDTH
+netMsg.getOffset();
```

De manera que la resta de 6.
Así el test 8 es exitoso, mientras que los restantes tests mantienen sus resultados iniciales.

Clases Modificadas: IPv6Protocol

2.2 Test 1.11 - Next Header Zero

2.2.1 Propósito

Este test se verifica que un nodo, genera una respuesta apropiada en caso de recibir un paquete con el campo Next Header en 0.

2.2.2 Procedimiento

1. TN transmite un paquete al NUT, que contiene un cabezal Hop-by-Hop Options header con el campo Next Header en 0.

2.2.3 Resultados esperados

2. El NUT debe enviar un mensaje ICMPv6 Parameter Problem al TN.
- El campo ICMPv6 Code debe ser 1(Unrecognized Next Header type).
- El campo ICMPv6 Pointer debe ser 0x28(offset del campo Next Header del cabezal Hop-by-Hop Options).

El NUT debe descartar el Echo Request y no enviar un Echo Reply al TN.

2.2.4 Resultado obtenido

Se recibió un Echo Reply en lugar de un ICMP de Error.

2.2.5 Solución

El error se debe a que si llega un NextHeader = 0 dentro de un Hop-by-Hop Options Header, se debería descartar el paquete pero no se hace.

Se modificó la clase HopByHopHeader de manera que si encuentra un 0 en el campo next Header, se lance una excepción UnexpectedNextHeaderProblemException la cual se maneja por la clase IPv6Protocol, la cual genera el paquete de error requerido.

```
int nextheader = buffer[offset-1] & 0xff;
if (nextheader == 0) {
    throw new UnrecognizedNextHeaderException(offset-1,
                            IPv6NextHeadersType.HOP_BY_HOP);
}
```

Clases modificadas:
`ip4jvm.net.protocols.IPv6Protocol`
`ip4jvm.net.protocols.headers.ipv6.HopByHopHeader`

Clase Creada:
`ip4jvm.net.exceptions.UnexpectedNextHeaderProblemException`

2.3 Test 1.16 - Destination Options Header precedes Fragment Header, Error from Fragment Header

2.3.1 Propósito

Verificar que un nodo procesa los cabezales de un paquete IPv6 en el orden correcto.

2.3.2 Procedimiento

1. TN transmite un paquete, que tiene un Hop-by-Hop Options header, un Destination Options header, y un Fragment header, en ese orden.
El Destination Options header tiene un Option Type 7 (desconocido).
El cabezal IPv6 tiene Payload Length que no es un múltiplo de 8 octetos, y el Fragment header tiene el M-bit seteado.

2.3.3 Resultados esperados

2. El NUT debería enviar un mensaje ICMPv6 Parameter Problem al TN.
- El campo Code debe ser 0 (erroneous header field).
- El campo Pointer debe ser 0x04 (offset del campo Payload Length).

El NUT debe descartar el Echo Request enviado desde el TN.

2.3.4 Resultado obtenido

Cannot receive ICMP Error message
No se envió un ICMP de error, ni ningún otro tipo de mensaje.

2.3.5 Solución

Ver test 1.53.

2.4 Test 1.17 - Fragment Header precedes Destination Options Header, Error from Fragment Header

2.4.1 Propósito

Verificar que un nodo procesa los cabezales de un paquete IPv6 en el orden correcto.

2.4.2 Procedimiento

1. TN transmite un Echo Request que tiene un Hop-by-Hop Options header, un Fragment header, y un Destination Options header en ese orden.
El cabezal IPv6 tiene un campo Payload Length que no es un múltiplo de 8, y un Fragment header que tiene el M-bit seteado.
El Destination Options header tiene un Option Type 135 (desconocido).

2.4.3 Resultados esperados

El NUT debe enviar un ICMPv6 Parameter Problem message al TN.
- El campo Code debe ser 0 (erroneous header field encountered).
- El campo Pointer debe ser 0x04 (offset del campo Payload Length).

El NUT debe descartar el Echo Request que viene del TN.

2.4.4 Resultado obtenido

Cannot receive ICMP Error message

No se envió un ICMP de error, ni ningún otro tipo de mensaje
Se lanza una Exception dentro de ip4jvm (Array index out of bounds)

2.4.5 Solución

Ver test 1.18.

2.5 Test 1.18 - Fragment Header precedes Destination Options Header, Error from Destination Options Header

2.5.1 Propósito

Verificar que un nodo procesa los cabezales de un paquete IPv6 en el orden correcto.

2.5.2 Procedimiento

El TN transmite un Echo Request que tiene un Hop-by-Hop Options header, un Fragment header, y un Destination Options header en ese orden.
El cabezal IPv6 tiene un Payload Length que no es un múltiplo de 8 octetos, y el Fragment header no tiene el M-bit seteado.
El Destination Options header tiene un Option Type desconocido (135).

2.5.3 Resultados esperados

El NUT debe enviar un ICMPv6 Parameter Problem message al TN.
El campo Code debe ser 2(unrecognized IPv6 Option encountered).
Si el IPv6 Parameter Problem message incluye un Fragment Header, el campo Pointer = 0x3A (offset del campo Option type del Destination Options header); sino el campo Pointer = 0x32 (offset del campo Option type del Destination Options header).
El NUT debe descartar el Echo Request enviado del TN.

2.5.4 Resultado obtenido

Cannot receive ICMP Error message

No se envió un ICMP de error, ni ningún otro tipo de mensaje, se lanzó una excepción.
IPv6ExtraHeader: línea 24 (java.lang.ArrayIndexOutOfBoundsException)

2.5.5 Solución

La excepción lanzada apunta a la clase
`ip4jvm.net.protocols.headers.ipv6.IPv6ExtraHeader`:

Método
`read(IPv6ProtocolHeader header, byte[] buffer, int offset)`

Línea 24:
```
nextHeader = buffer[offset] & 0xff;
```

Evidentemente se quiere acceder a una posición del paquete no existente. Esto se debe a que `offset` se encuentra mal seteado. Se modificó la forma en que se setea `offset`:

```
ip4jvm.net.protocols.headers.IPv6ProtocolHeader
readExtraHeaders(byte[] message, int offset)

if (eH.getType()==IPv6NextHeadersType.FRAGMENT_HEADER)
      offset += 8;
else
      offset += eH.getLength();
```

2.6 Test 1.39 - Unrecognized Routing Type 0

2.6.1 Propósito

Verificar que un nodo procesa correctamente un paquete IPv6 siendo un nodo intermedio que contiene un Routing header con un Routing Type inválido (0). Por el RFC 5095, el routing type 0 no es válido.

2.6.2 Procedimiento

TN envía un Echo Request que tiene un Routing header con el campo Routing Type en 0 y con el campo Segments Left en 1. El Echo Request está destinado para el NUT.

2.6.3 Resultados esperados

El NUT debe descartar el Echo Request y enviar un ICMP Parameter Problem Code 0 message al Global Address del TN. El campo Pointer debe ser 0x2A (offset del campo Routing Type del Routing header).

2.6.4 Resultado obtenido

Cannot receive ICMPv6 Parameter Problem message
En lugar de un ICMP Error, se envió un Echo Request.

2.6.5 Solución

Ver test 1.41.

2.7 Test 1.41 - Unrecognized Routing Type 0

2.7.1 Propósito

Verificar que un nodo procesa correctamente un paquete IPv6 siendo un nodo intermedio que contiene un Routing header con un Routing Type inválido (0). De acuerdo con el RFC 5095, el routing type 0 no es válido.

2.7.2 Procedimiento

TN envía un Echo Request que tiene un Routing header con el campo Routing Type en 0 y con el campo Segments Left en 1. El Echo Request está destinado para el NUT.

2.7.3 Resultados esperados

El NUT debe descartar el Echo Request y enviar un ICMP Parameter Problem Code 0 message al Global Address del TN.

El campo Pointer debe ser 0x2A (offset del campo Routing Type del Routing header).

2.7.4 Resultado obtenido

Cannot receive ICMPv6 Parameter Problem message
En lugar de un ICMP Error, se envía un Echo Request.

2.7.5 Solución

Ya que por el RFC 5095, se elimina el uso de los Routing Headers con Routing Type = 0, se modificó la clase que trata con este tipo de headers.

`ip4jvm.net.protocols.extras.ipv6ExtraHeadPrcsrs.IPv6RouterHeadPrcsr`

Se cambió toda la sección que manejaba los routing headers
`IPv6RoutingHeadersType.TYPE0_ROUTING_HEADER`

Por

```
int offset = rh.getPosition();
toSend = new ICMPv6PProblem(ICMPv6PProblem.ERRONEOUS_HEADER_FIELD,
offset+rh.NEXT_HEAD_WIDTH+rh.LENGTH_WIDTH-ns.getMessage().getOffset(),
ns.getMessage().getBytesfromOffset());
sendICMP(toSend, ns.getNetconfig(), dst);
```

Este código envía el ICMP de error que se requiere de parte del test.

2.8 Test 1.44 - Fragment IDs Differ Between Fragments

2.8.1 Propósito

Verificar que un nodo re ensambla correctamente paquetes fragmentados y distingue entre diferentes fragmentos de paquetes usando el Source Address, Destination Address y Fragment ID.

2.8.2 Procedimiento

TN transmite los fragmentos 1, 2 y 3 en orden.
El 1er y el 3er fragmento tienen un Fragment ID = 2999.
El 2do fragmento tiene un Fragment ID = 3000.
El Source y el Destination Address de todos los fragmentos es el mismo.

2.8.3 Resultados esperados

El NUT no debe transmitir un Echo Reply al TN, ya que el Echo Request no pudo ser re ensamblado por diferencias en el Fragment ID. El NUT debe transmitir un ICMPv6 Time Exceeded Message al TN 60 segundos después de la recepción del 1er fragmento.

2.8.4 Resultado obtenido

Exceed too fast (50 sec.)

Se envía el Time Exceeded error 15 segundos antes de lo esperado

2.8.5 Solución

Ver test 1.51

2.9 Test 1.45 - Source Addresses Differ Between Fragments

2.9.1 Propósito

Verificar que un nodo re ensambla correctamente paquetes fragmentados y distingue entre diferentes fragmentos de paquetes usando el Source Address, Destination Address y Fragment ID.

2.9.2 Procedimiento

TN transmite los fragmentos 1, 2 y 3 en orden.
El 1er y el 3er fragmento tienen Source Address = Dirección local del TN.
El 2do fragmento tiene un Source Address diferente.
El Fragment ID y el Destination Address de todos los fragmentos es el mismo.

2.9.3 Resultados esperados

El NUT no debe transmitir un Echo Reply al TN, ya que el Echo Request no pudo ser re ensamblado por diferencias en el Source Address. El NUT debe transmitir un ICMPv6 Time Exceeded Message al TN 60 segundos después de la recepción del 1er fragmento.

2.9.4 Resultado obtenido

Exceed too fast (50 sec.)

Se envía el Time Exceeded error 15 segundos antes de lo esperado

2.9.5 Solución

Ver test 1.51

2.10 Test 1.46 - Destination Address Differ Between Fragments

2.10.1 Propósito

Verificar que un nodo re ensambla correctamente paquetes fragmentados y distingue entre diferentes fragmentos de paquetes usando el Source Address, Destination Address y Fragment ID.

2.10.2 Procedimiento

TN transmite los fragmentos 1, 2 y 3 en orden.
El 1er y el 3er fragmento tienen Destination Address = Dirección local del NUT.
El 2do fragmento tiene un Destination Address diferente.
El Fragment ID y el Source Address de todos los fragmentos es el mismo.

2.10.3 Resultados esperados

El NUT no debe transmitir un Echo Reply al TN, ya que el Echo Request no pudo ser re ensamblado por diferencias en el Destination Address. El NUT debe transmitir un ICMPv6 Time Exceeded Message al TN 60 segundos después de la recepción del 1er fragmento.

2.10.4 Resultado obtenido

Exceed too fast (50 sec.)

Se envía el Time Exceeded error 15 segundos antes de lo esperado

2.10.5 Solución

Ver test 1.51

2.11 Test 1.48 - Time Elapsed Between Fragments less than Sixty Seconds

2.11.1 Propósito

Verificar que un nodo toma las acciones apropiadas cuando el tiempo de re ensamblado se ha excedido para un paquete.

2.11.2 Procedimiento

TN transmite los fragmentos 1, 2 y 3 en orden.
Hay un atraso de 55 segundos entre la transmision del 1er fragmento; y el 2do y 3ro.

2.11.3 Resultados esperados

El NUT debe transmitir un Echo Reply al TN en respuesta al Echo Request re ensamblado. (Esto se debe a que el 2do y 3er fragmento llegan antes del timeout).

2.11.4 Resultado obtenido

Exceed too fast (50 sec.)

Se envía el Time Exceeded error 15 segundos antes de lo esperado

2.11.5 Solución

Ver test 1.51

2.12 Test 1.49 - Time Exceeded Before Last Fragments Arrive

2.12.1 Propósito

Verificar que un nodo toma las acciones apropiadas cuando el tiempo de re ensamblado se ha excedido para un paquete.

2.12.2 Procedimiento

TN transmite los fragmentos 1, 2 y 3 en orden.
Hay un atraso de 65 segundos entre la transmision del 1er fragmento; y el 2do y 3ro.

2.12.3 Resultados esperados

El NUT no debe transmitir un Echo Reply al TN ya que el Echo Request no pudo ser re ensamblado. El NUT debe enviar un ICMPv6 Time Exceeded Message al TN 60 segundos después de la recepción del 1er fragmento.

El campo Code debe ser 1(Fragment Reassembly Time Exceeded).
El campo Unused debe ser inicializado a 0.
El Source Address debe ser el mismo que el Destination Address del Echo Request enviado por el TN.
El Destination Address debe ser el mismo que el Source Address del Echo Request enviado por el TN.

2.12.4 Resultado obtenido

Exceed too fast (50 sec.)

Se envía el Time Exceeded error 15 segundos antes de lo esperado

2.12.5 Solución

Ver test 1.51

2.13 Test 1.50 - Time Exceeded (Global), Only First Fragment Received

2.13.1 Propósito

Verificar que un nodo toma las acciones apropiadas cuando el tiempo de re ensamblado se ha excedido para un paquete.

2.13.2 Procedimiento

TN únicamente transmite el 1er fragmento de un paquete.

2.13.3 Resultados esperados

El NUT no debe transmitir un Echo Reply al TN, ya que el Echo Request no pudo ser completado.

El NUT debe enviar un ICMPv6 Time Exceeded Message al TN 60 segundos después de la recepción del 1er fragmento.

El campo Code debe ser 1(Fragment Reassembly Time Exceeded).
El campo Unused debe ser inicializado a 0.
El Source Address debe ser el mismo que el Destination Address del Echo Request enviado por el TN.
El Destination Address debe ser el mismo que el Source Address del Echo Request enviado por el TN.

2.13.4 Resultado obtenido

Exceed too fast (50 sec.)

Se envía el Time Exceeded error 15 segundos antes de lo esperado

2.13.5 Solución

Ver test 1.51

2.14 Test 1.51 - Time Exceeded (Link-local), Only First Fragment Received

2.14.1 Propósito

Verificar que un nodo toma las acciones apropiadas cuando el tiempo de re ensamblado se ha excedido para un paquete.

2.14.2 Procedimiento

TN únicamente transmite el 1er fragmento de un paquete.
Source Address = Link-local address de TN
Destination Address = Link-local address del NUT.

2.14.3 Resultados esperados

El NUT no debe transmitir un Echo Reply al TN, ya que el Echo Request no fue completado.
El NUT debe enviar un ICMPv6 Time Exceeded Message al TN 60 segundos después de la recepción del 1er fragmento.
El campo Code debe ser 1 (Fragment Reassembly Time Exceeded).
El campo Unused debe ser inicializado a 0.
El Source Address debe ser el mismo que el Destination Address del Echo Request enviado por el TN.
El Destination Address debe ser el mismo que el Source Address del Echo Request enviado por el TN.

2.14.4 Resultado obtenido

Exceed too fast (50 sec.)

Se envía el Time Exceeded error 15 segundos antes de lo esperado

2.14.5 Solución

Configurar el archivo $TAHI_TESTS/spec.p2/config.pl con los valores

```
$exceed_max             = 50;
$exceed_min             = 40;
```

2.15 Test 1.53 - Fragment Header M-Bit Set, Payload Length Invalid

2.15.1 Propósito

Verificar que un nodo actúa apropiadamente cuando recibe un fragmento de un paquete con el M-bit seteado (o sea que el paquete tiene más fragmentos), pero que tiene el campo Payload Length con un valor que no es un múltiplo de 8 bytes.

2.15.2 Procedimiento

TN envía un Echo Request que tiene un Fragment header con el M-bit seteado. El campo Payload Length tiene el valor 21, que no es múltiplo de 8.

2.15.3 Resultados esperados

El NUT no debe enviar un Echo Reply al TN, ya que el fragmento debe ser descartado.
El NUT debe enviar un ICMPv6 Parameter Problem message al TN.
El campo Code debe ser 0 (erroneous header field encountered).
El campo Pointer debe ser 0x04 (offset del campo Payload Length en el cabezal IPv6).

2.15.4 Resultado obtenido

Cannot receive ICMP Error message.

2.15.5 Solución

Para resolver este error, se procedió de la siguiente manera:
Si M bit está en uno, hay que ver si Payload Length es un múltiplo de 8. En caso afirmativo se procesa el paquete como es previsto y en caso negativo se lanza una excepción.

Clase modificada:
ip4jvm.net.protocols.extras.ipv6ExtraHeadPrcsrs.IPv6FragmentHeadPrcsr

```
int payloadLength = head.getPayloadLength();
FragmentHeader fragHead= (FragmentHeader) extraHead;

if (fragHead.getMFlag() && payloadLength %8 !=0){
      ICMPv6PProblem toSend  = new
ICMPv6PProblem(ICMPv6PProblem.ERRONEOUS_HEADER_FIELD, 18-
ns.getMessage().getOffset(), ns.getMessage().getBytesfromOffset());
      sendICMP(toSend, ns.getNetconfig(), head.getSourceAddress());
      return false;
   }
```

3 Section II: RFC 4861 - Neighbor Discovery for IPv6

Los siguientes tests cubren la especificación del protocolo de Neighbor Discovery (ND) para IPv6, como se describe en el RFC 4861.
Este protocolo consiste en un mecanismo con el cual un nodo que se acaba de incorporar a una red, descubre la presencia de otros nodos en el mismo enlace, además de ver sus direcciones IP. Este protocolo también se ocupa de mantener limpios los caches donde se almacena la información relativa al contexto de la red a la que está conectado un nodo. Así cuando una ruta hacia un cierto nodo falla, el router correspondiente buscará rutas alternativas. Emplea los mensajes de ICMPv6, y es la base para permitir el mecanismo de autoconfiguración en IPv6.
Los mensajes utilizados por el mecanismo de ND son:

- RS – Router Solicitation: es generado por una interfaz cuando ésta es activada, para pedir a los nodos que se anuncien.

- RA – Router Advertisement : es producido por los nodos periódicamente (entre cada 4 y 1800 segundos), o bien se produce por una "solicitud de router", de esta manera informa de su presencia así como de otros parámetros de enlace y de Internet, como prefijos (uno o varios), tiempos de vida y configuración de direcciones.

- NS – Neighbor solicitation: lo generan los nodos para determinar la dirección en la capa de enlace de sus vecinos, o para asegurarse de que el nodo vecino es alcanzable, aunque también se genera para detectar las direcciones IP duplicadas.

- NA – Neighbor Advertisement: los nodos lo producen como respuesta a la "solicitud de vecino", principalmente, aunque también para indicar cambios de direcciones en el nivel de enlace.

- Redirect: los nodos generan este paquete para informar a los enrutadores de que existe una ruta mejor para llegar a un determinado destino. Es equivalente, en parte a "ICMP redirect".

Estado inicial:

1	PASS	21	PASS	41	FAIL	61	FAIL	81	PASS	101	FAIL
2	FAIL	22	PASS	42	PASS	62	FAIL	82	FAIL	102	FAIL
3	PASS	23	PASS	43	PASS	63	FAIL	83	FAIL	103	FAIL
4	PASS	24	PASS	44	PASS	64	FAIL	84	FAIL	104	FAIL
5	FAIL	25	FAIL	45	FAIL	65	FAIL	85	PASS	105	FAIL
6	FAIL	26	PASS	46	FAIL	66	FAIL	86	FAIL	106	FAIL
7	PASS	27	FAIL	47	FAIL	67	PASS	87	FAIL	107	FAIL
8	PASS	28	FAIL	48	FAIL	68	FAIL	88	FAIL	108	FAIL
9	PASS	29	FAIL	49	FAIL	69	FAIL	89	PASS	109	FAIL
10	FAIL	30	FAIL	50	FAIL	70	FAIL	90	FAIL	110	FAIL
11	FAIL	31	FAIL	51	FAIL	71	PASS	91	FAIL	111	FAIL
12	FAIL	32	PASS	52	FAIL	72	FAIL	92	FAIL	112	FAIL
13	FAIL	33	PASS	53	PASS	73	FAIL	93	PASS	113	FAIL
14	FAIL	34	FAIL	54	FAIL	74	FAIL	94	FAIL	114	FAIL
15	FAIL	35	FAIL	55	PASS	75	PASS	95	PASS	115	FAIL
16	PASS	36	PASS	56	FAIL	76	FAIL	96	FAIL	116	FAIL
17	PASS	37	FAIL	57	FAIL	77	PASS	97	PASS	117	FAIL
18	PASS	38	FAIL	58	FAIL	78	FAIL	98	FAIL	118	FAIL
19	PASS	39	FAIL	59	FAIL	79	FAIL	99	PASS	119	FAIL
20	PASS	40	FAIL	60	FAIL	80	FAIL	100	FAIL	120	FAIL

121	FAIL	141	PASS	161	FAIL	181	FAIL	201	FAIL	221	FAIL
122	FAIL	142	FAIL	162	FAIL	182	FAIL	202	FAIL	222	FAIL
123	FAIL	143	PASS	163	FAIL	183	FAIL	203	FAIL	223	FAIL
124	FAIL	144	FAIL	164	PASS	184	FAIL	204	FAIL	224	FAIL
125	FAIL	145	FAIL	165	FAIL	185	FAIL	205	FAIL	225	FAIL
126	FAIL	146	FAIL	166	PASS	186	FAIL	206	FAIL	226	FAIL
127	PASS	147	PASS	167	PASS	187	FAIL	207	FAIL	227	FAIL
128	FAIL	148	PASS	168	PASS	188	FAIL	208	FAIL	228	FAIL
129	FAIL	149	PASS	169	FAIL	189	FAIL	209	FAIL	229	FAIL
130	FAIL	150	FAIL	170	FAIL	190	FAIL	210	FAIL	230	FAIL
131	FAIL	151	FAIL	171	FAIL	191	FAIL	211	FAIL	231	FAIL
132	FAIL	152	PASS	172	FAIL	192	FAIL	212	FAIL	232	FAIL
133	FAIL	153	FAIL	173	FAIL	193	FAIL	213	FAIL	233	FAIL
134	FAIL	154	PASS	174	FAIL	194	FAIL	214	FAIL	234	FAIL
135	FAIL	155	FAIL	175	FAIL	195	FAIL	215	FAIL	235	FAIL
136	FAIL	156	PASS	176	FAIL	196	FAIL	216	FAIL	236	FAIL
137	FAIL	157	FAIL	177	FAIL	197	FAIL	217	FAIL		
138	FAIL	158	FAIL	178	FAIL	198	FAIL	218	FAIL		
139	PASS	159	FAIL	179	FAIL	199	FAIL	219	FAIL		
140	FAIL	160	PASS	180	FAIL	200	FAIL	220	FAIL		

Inicialmente fallaron 186 tests, aunque podemos excluir el test 2.2 que simplemente falló debido a que no se reiniciaba el stack.

3.1 Test 2.5 - Part A: Single Queue

3.1.1 Propósito

Verificar que un nodo encola apropiadamente paquetes mientras espera por el Address resolution del Next hop.

3.1.2 Procedimiento

1. TN envía un Echo Request (A), 3 veces. El Sequence number se incrementa cada vez.
3. TN envía un Neighbor Advertisement (NA) en respuesta a cualquier Neighbor Solicitation (NS) del NUT.

3.1.3 Resultados esperados

2. El NUT debe enviar un Neighbor Solicitation con Target Address = -local address del TN. El NUT debe enviar Echo Replies a TN en respuesta al Echo Request (A).
4. Los Echo Reply deben corresponder a los últimos 3 Echo Request enviados por el TN al NUT, indicando éxito en el encolamiento de paquetes mientras espera por el Address resolution. El número de Echo Reply DEBE SER AL MENOS 1.

3.1.4 Resultado obtenido

Couldn't observe NS

Se recibieron 3 Echo Reply en lugar de un NS (error en paso 2).

3.1.5 Solución

Ver test 2.11

3.2 Test 2.6 - Part B: Multiple Queues

3.2.1 Propósito

Verificar que un nodo encola apropiadamente paquetes mientras espera por el Address resolution del Next hop.

3.2.2 Procedimiento

1. TN1 envía un paquete Echo Request (A), 3 veces. El Sequence number se incrementa cada vez.
2. TN2 envía un paquete Echo Request (B), 4 veces. El Sequence number se incrementa cada vez.
4. TN1 y TN2 envían un NA en respuesta a cualquier NS del NUT.

3.2.3 Resultados esperados

3. El NUT debe enviar un NS con Target Address = TN1's link-local address. El NUT debe enviar Echo Replies a TN1 en respuesta al paquete A. El NUT debe enviar un NS con Target Address = TN2's link-local address. El NUT debe enviar Echo Replies a TN2 en respuesta al paquete B.
5. Los Echo Reply deben corresponder a los últimos 3 Echo Request enviados por el TN1 al NUT, indicando éxito en el encolamiento de paquetes mientras espera por el Address resolution. El número de Echo Reply DEBE SER NO MENOS DE 1.
Los Echo Reply deben corresponder a los últimos 4 Echo Request enviados por el TN2 al NUT, indicando éxito en el encolamiento de paquetes mientras espera por el Address resolution. El número de Echo Reply DEBE SER AL MENOS 1.

3.2.4 Resultado obtenido

Couldn't observe NS for TN

El test falló tras el paso 2 ya que los NS para TN1 se enviaron cada uno con 4 segundos de diferencia en lugar de 1 como se esperaba.

3.2.5 Solución

Ver test 2.11

3.3 Test 2.10 - Part A: Neighbor Solicitation Origination, Target Address Being Link-local

3.3.1 Propósito

Verificar que un nodo genera apropiadamente Neighbor Solicitations (NS) mientras intenta resolver la dirección de un vecino (neighbor).

3.3.2 Procedimiento

1. Setear el Retransmit Interval a 1 segundo.
2. TN1 envía el paquete A.
Source address = TN1's link-local address
Destination address = Link-local address del NUT.
4. Repetir 1 y 2 con un Retransmit Interval de 5 segundos.

3.3.3 Resultados esperados

3. El NUT envía NS's (con Target Address = TN1's Link-local Address) cada 1 segundo. El NUT no debe enviar más de un NS por segundo. Cada uno debe tener la opción Source Link-Layer Address (SLLA). El número máximo de NS's debe ser MAX_MULTICAST_SOLICIT, que debe ser 3.

5. El NUT envía NS's (con Target Address = TN1's Link-local Address) cada 5 segundos. El NUT no debe enviar más de un NS cada 5 segundos. Cada uno debe tener la opción Source Link-Layer Address (SLLA). El número máximo de NS's debe ser MAX_MULTICAST_SOLICIT, que debe ser 3.

3.3.4 Resultado obtenido

Couldn't observe NS

Los NS's se enviaron cada 4 segundos

3.3.5 Solución

Ver test 2.11

3.4 Test 2.11 - Part B: Neighbor Solicitation Origination, Target Address Being Global

3.4.1 Propósito

Verificar que un nodo genera apropiadamente Neighbor Solicitations (NS) mientras intenta resolver la dirección de un vecino (neighbor).

3.4.2 Procedimiento

1. Setear el Retransmit Interval a 1 segundo.
2. TN1 envía el paquete A.
Source address = Global address del TN
Destination address = Global address del NUT.
4. Repetir 1 y 2 con un Retransmit Interval de 5 segundos.

3.4.3 Resultados esperados

3. El NUT envía NS's (con Target Address = Global address del TN) cada 1 segundo. El NUT no debe enviar más de un NS por segundo. Cada uno debe tener la opción Source Link-Layer Address (SLLA). El número máximo de NS's debe ser MAX_MULTICAST_SOLICIT, que debe ser 3.
5. El NUT envía NS's (con Target Address = Global address del TN) cada 5 segundos. El NUT no debe enviar más de un NS cada 5 segundos. Cada uno debe tener la opción Source Link-Layer Address (SLLA). El número máximo de NS's debe ser MAX_MULTICAST_SOLICIT, que debe ser 3.

3.4.4 Resultado obtenido

Couldn't observe NS

Los NS's se enviaron cada 4 segundos

3.4.5 Solución

El error en estos tests consistía en que se enviaban NS's cada 4 segundos

Clase:
`ip4jvm.net.applications.NeighborDiscovery`
Método:
`setState(Inet6Address address, int state, IPv6NetConfig source)`

La siguiente línea de código:

```
t.scheduleAtFixedRate(stateChanger,cantSegs,
                      RTR_SOLICITATION_INTERVAL);
```

Envía NS's cada *RTR_SOLICITATION_INTERVAL = 4000;*

Esto es incorrecto ya que se debe usar el valor de Retransmit Timer (1000 ms por defecto)

Por lo tanto se cambio el código antes mencionado por el siguiente:

```
long retransTimer = ((IPv6NetParameters)stack.getNetParameters()).
                    getRetransmitTimer(source.getJth());
if (retransTimer == 0)
      retransTimer = RETRANS_TIMER;
t.scheduleAtFixedRate(stateChanger, 1, retransTimer);
break;
```

3.5 Test 2.12 - Part A: Neighbor Solicitation Origination, Link-local => Link-local

3.5.1 Propósito

Verificar que un nodo genera apropiadamente Neighbor Solicitations (NS) mientras intenta resolver la dirección de un vecino (neighbor).

3.5.2 Procedimiento

1. TN1 envía un paquete A (Source address = TN1's link-local address y Destination address = Link-local address del NUT).
2. TN1 envía un NA al recibir un NS del NUT.
3. Esperar REACHABLE_TIME * MAX_RANDOM_FACTOR segundos para que el NCE de TN1 pase a estado STALE.
4. TN1 envía el paquete A. (Source address = TN1's link-local address y Destination address = Link-local address del NUT).
6. Esperar DELAY_FIRST_PROBE_TIME segundos para que el NCE de TN1 pase al estado PROBE.

3.5.3 Resultados esperados

2. En respuesta al paquete A, el NUT debe enviar NS's donde Target Address = TN1's link-local Address a intervalos de 1 segundo. El NUT debe enviar no más de 1 NS por segundo. Cuando se recibe el NA de TN1, el NUT debe enviar un Echo Reply en respuesta al paquete A. El NCE de TN1 está en estado REACHABLE.
5. En respuesta al paquete A, el NUT debe enviar un Echo Reply.
7. El NUT debe enviar NS's donde Source address = Link-local address del NUT y Destination address = TN1's link-local address. El número máximo de NS's que el NUT puede enviar es 3.

3.5.4 Resultado obtenido

Couldn't observe NS.

Se recibió un Echo Reply, en lugar de un NS, los 3 NS se recibieron con atraso.

3.5.5 Solución

Ver Test 2.15

3.6 Test 2.13 - Part B: Neighbor Solicitation Origination, Global => Global

3.6.1 Propósito

Verificar que un nodo genera apropiadamente Neighbor Solicitations (NS) mientras intenta resolver la dirección de un vecino (neighbor).

3.6.2 Procedimiento

1. TN1 envía un paquete B (Source address = Global address del TN y Destination address = Global address del NUT).
2. TN1 envía un NA al recibir un NS del NUT.
3. Esperar REACHABLE_TIME * MAX_RANDOM_FACTOR segundos para que el NCE de TN1 pase a estado STALE.
4. TN1 envía el paquete B. (Source address = Global address del TN y Destination address = Global address del NUT).
6. Esperar DELAY_FIRST_PROBE_TIME segundos para que el NCE de TN1 pase al estado PROBE.

3.6.3 Resultados esperados

2. En respuesta al paquete B, el NUT debe enviar NS's donde Target Address = Global address del TN a intervalos de 1 segundo. El NUT debe enviar no más de 1 NS por segundo. Cuando se recibe el NA de TN1, el NUT debe enviar un Echo Reply en respuesta al paquete B. El NCE de TN1 está en estado REACHABLE.
5. En respuesta al paquete B, el NUT debe enviar un Echo Reply.
7. El NUT debe enviar NS's donde Source address = Global address del NUT y Destination address = Global address del TN. El número máximo de NS's que el NUT puede enviar es 3.

3.6.4 Resultado obtenido

Couldn't observe NS.

Se recibió un Echo Reply, en lugar de un NS, los 3 NS se recibieron con atraso.

3.6.5 Solución

Ver test 2.15

3.7 Test 2.14 - Part C: Neighbor Solicitation Origination, Link-local => Global

3.7.1 Propósito

Verificar que un nodo genera apropiadamente Neighbor Solicitations (NS) mientras intenta resolver la dirección de un vecino (neighbor).

3.7.2 Procedimiento

1. TN1 envía un paquete C (Source address = TN1's Link-local address y Destination address = Global address del NUT).
2. TN1 envía un NA al recibir un NS del NUT.
3. Esperar REACHABLE_TIME * MAX_RANDOM_FACTOR segundos para que el NCE de TN1 pase a estado STALE.
4. TN1 envía el paquete C. (Source address = TN1's Link-local address y Destination address = Global address del NUT).
6. Esperar DELAY_FIRST_PROBE_TIME segundos para que el NCE de TN1 pase al estado PROBE.

3.7.3 Resultados esperados

2. En respuesta al paquete C, el NUT debe enviar NS's donde Target Address = TN1's Link-local Address a intervalos de 1 segundo. El NUT debe enviar no más de 1 NS por segundo. Cuando se recibe el NA de TN1, el NUT debe enviar un Echo Reply en respuesta al paquete C. El NCE de TN1 está en estado REACHABLE.
5. En respuesta al paquete C, el NUT debe enviar un Echo Reply.
7. El NUT debe enviar NS's donde Source address = Global address del NUT ó Link-Local address y Destination address = TN1's Link-local address. El número máximo de NS's que el NUT puede enviar es 3.

3.7.4 Resultado obtenido

Observed too less NSs

Se debían recibir 3 NS's pero se recibió uno solo, seguido de un NA.

3.7.5 Solución

Ver test 2.15

3.8 Test 2.15 - Part D: Neighbor Solicitation Origination, Global => Link-local

3.8.1 Propósito

Verificar que un nodo genera apropiadamente Neighbor Solicitations (NS) mientras intenta resolver la dirección de un vecino (neighbor).

3.8.2 Procedimiento

1. TN1 envía un paquete D (Source address = Global address del TN y Destination address = Link-local address del NUT).
2. TN1 envía un NA al recibir un NS del NUT.
3. Esperar REACHABLE_TIME * MAX_RANDOM_FACTOR segundos para que el NCE de TN1 pase a estado STALE.
4. TN1 envía el paquete D. (Source address = Global address del TN y Destination address = Link-local address del NUT).
6. Esperar DELAY_FIRST_PROBE_TIME segundos para que el NCE de TN1 pase al estado PROBE.

3.8.3 Resultados esperados

2. En respuesta al paquete D, el NUT debe enviar NS's donde Target Address = Global address del TN a intervalos de 1 segundo. El NUT debe enviar no más de 1 NS por segundo. Cuando se recibe el NA de TN1, el NUT debe enviar un Echo Reply en respuesta al paquete D. El NCE de TN1 está en estado REACHABLE.
5. En respuesta al paquete C, el NUT debe enviar un Echo Reply.
7. El NUT debe enviar NS's donde Source address = Global address del NUT ó Link-Local address y Destination address = Global address del TN. El número máximo de NS's que el NUT puede enviar es 3.

3.8.4 Resultado obtenido

Observed too less NSs

Se debían recibir 3 NS's pero se recibió uno solo, seguido de un NA.

3.8.5 Solución

La solución fue aparte de la dada en el Test 2.11 fue modificar la clase:
`ip4jvm.net.applications.NeighborDiscovery`

Método:
`rcvICMPv6NeighborAdvertisement(ICMPv6Header, IPv6ProtocolHeader, IPv6NetParameters, String, NetConfigs)`

Más precisamente agregar el siguiente código cuando se chequea la siguiente condición

```
a!=null && entryState == NeighborCache.STATE_INCOMPLETE && solicited
```

Setear el NCE a REACHABLE:

```
setState(icmpNA.targetAddress,NeighborCache.STATE_REACHABLE,
(IPv6NetConfig) netConfig.getFirst());
```

3.9 Test 2.25 - Part B: Multicast Neighbor Solicitation

3.9.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe Neighbor Solicitations y no existe un NCE (Neighbor Cache Entry) para ese vecino (neighbor).

3.9.2 Procedimiento

1. TN1 envía un NS (B).
2. TN1 envía un Echo Request al NUT.

3.9.3 Resultados esperados

1. El NUT crea un NCE para TN1 y setea el estado de la entrada a STALE. El NUT responde al NS B enviando un NA.

Luego de responder al NS, el NUT debe responder al Echo Request enviando un Echo Reply pasando el estado del NCE a DELAY. Una vez transcurridos DELAY_FIRST_PROBE_TIME milisegundos, el NUT debe enviar un NS a TN1.

3.9.4 Resultado obtenido

Couldn't observe NS

3.9.5 Solución

Ver test 2.11

3.10 Test 2.27 - Part A: Unicast Neighbor Solicitation

3.10.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE (Neighbor Cache Entry) para ese vecino (neighbor) está en estado INCOMPLETE.

3.10.2 Procedimiento

1. TN1 envía un Echo Request (A).
3. TN1 envía un NS (B).
5. TN1 envía un Echo Request al NUT.

3.10.3 Resultados esperados

2. El NUT debe crear un NCE para TN1 y setear su estado a INCOMPLETE. El NUT debe enviar un multicast NS a TN1.
4. Tras recibir el NS de TN1, el NUT debe enviar el Echo Reply encolado a TN1. El NUT debe actualizar el NCE de TN1 a estado STALE and actualizar su Link-layer address. El NUT debe responder al NS B enviando un NA.
6. El NUT debe responder al Echo Request enviando un Echo Reply y setear el estado del NCE a DELAY. Pasados DELAY_FIRST_PROBE_TIME milisegundos, el NUT debe enviar un Unicast NS a TN1.

3.10.4 Resultado obtenido

Couldn't observe NS

Se recibió un Echo Reply en lugar de un NS.

3.10.5 Solución

Ver Test 2.28

3.11 Test 2.28 - Part B: Multicast Neighbor Solicitation

3.11.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE (Neighbor Cache Entry) para ese vecino (neighbor) está en estado INCOMPLETE.

3.11.2 Procedimiento

1. TN1 envía un Echo Request (A).
3. TN1 envía un NS (C).
5. TN1 envía un Echo Request al NUT.

3.11.3 Resultados esperados

2. El NUT debe crear un NCE para TN1 y setear su estado a INCOMPLETE. El NUT debe enviar un multicast NS a TN1.
4. Tras recibir el NS de TN1, el NUT debe enviar el Echo Reply encolado a TN1. El NUT debe actualizar el NCE de TN1 a estado STALE and actualizar su Link-layer address. El NUT debe responder al NS B enviando un NA.
6. El NUT debe responder al Echo Request enviando un Echo Reply y setear el estado del NCE a DELAY. Pasados DELAY_FIRST_PROBE_TIME milisegundos, el NUT debe enviar un Unicast NS a TN1.

3.11.4 Resultado obtenido

Couldn't observe NS

Falló el paso 2 ya que no se envió un multicast NS y se envió un Echo Reply en lugar de un NS.

3.11.5 Solución

Primero se deben aplicar las modificaciones descritas en los tests 2.11 y 2.15.

Arreglado esto, el error pasa al paso 4, ya que no se recibe el Echo Reply encolado que se describe ("*Tras recibir el NS de TN1, el NUT debe enviar el Echo Reply encolado a TN1.*").

Para esto hay que modificar cuando se recibe un NS en la clase:

`ip4jvm.net.applications.NeighborDiscovery`

Método;

`addNeighborHost(String, MACAddress, Inet6Address, boolean, int, boolean)`

Más precisamente modificar la siguiente pieza de código:

```
if (state!=NeighborCache.STATE_STALE){
      stack.activateJobs(2,ip,mac,jth);
}
```

Sacando el if, se envían los paquetes pendientes (el Echo Reply).

3.12 Test 2.29 - Part C: Unicast Neighbor Solicitation without SLL

3.12.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE (Neighbor Cache Entry) para ese vecino (neighbor) está en estado INCOMPLETE.

3.12.2 Procedimiento

1. TN1 envía un Echo Request (A).
3. TN1 envía un NS (B) que no tiene la opción SLLA (Source Link-Layer Address).
5. TN1 envía un Echo Request al NUT.

3.12.3 Resultados esperados

2. El NUT debe crear un NCE para TN1 y setear su estado a INCOMPLETE. El NUT debe enviar un multicast NS a TN1.
4. Tras recibir el NS de TN1, el NUT NO debe actualizar el NCE de TN1.

3.12.4 Resultado obtenido

Couldn't observe NS

Se recibió un Echo Reply en lugar de un NS.

3.12.5 Solución

Ver Test 2.11

3.13 Test 2.30 - Part A: Unicast Neighbor Solicitation with the same SLLA

3.13.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado REACHABLE.

3.13.2 Procedimiento

1. TN1 envía un Echo Request A.
3. TN1 envía un NA B.
5. TN1 envía un Echo Request A.
7. TN1 envía un NS C.
8. TN1 envía un Echo Request A.

3.13.3 Resultados esperados

2. El NUT crea un NCE para TN1 con estado INCOMPLETE. El NUT debe enviar un multicast NS a TN1.
4. Tras recibir un NA de TN1, el NUT debe enviar el Echo Reply encolado a TN1. El NUT debe actualizar el NCE de TN1 al estado REACHABLE y actualizar el Link-layer address para TN1.
6. Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.
9. El NUT NO debe actualizar el NCE de TN1, el NUT debe responder al NS C enviando un NA. Tras responder al NS, el NUT debe responder al Echo Request enviando un Echo Reply manteniéndose en estado REACHABLE. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.

3.13.4 Resultado obtenido

Couldn't observe NS

Se recibió un Echo Reply en lugar de un NS.

3.13.5 Solución

Ver Test 2.11

3.14 Test 2.31 - Part B: Unicast Neighbor Solicitation with a different SLLA

3.14.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado REACHABLE.

3.14.2 Procedimiento

1. TN1 envía un Echo Request A.
3. TN1 envía un NA B.
5. TN1 envía un Echo Request A.
7. TN1 envía un NS C con un SLLA diferente.
8. TN1 envía un Echo Request A.

3.14.3 Resultados esperados

2. El NUT crea un NCE para TN1 con estado INCOMPLETE. El NUT debe enviar un multicast NS a TN1.
4. Tras recibir un NA de TN1, el NUT debe enviar el Echo Reply encolado a TN1. El NUT debe actualizar el NCE de TN1 al estado REACHABLE y actualizar el Link-layer address para TN1.
6. Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.
9. El NUT debe actualizar el NCE de TN1 a estado STALE y actualizar el Link-layer address de TN1 con el valor recibido en el NS C, el NUT debe responder al NS C enviando un NA. Tras responder al NS, el NUT debe responder al Echo Request enviando un Echo Reply y pasar a estado DELAY. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1 donde Target Address = nuevo Link-layer address de TN1.

3.14.4 Resultado obtenido

Couldn't observe NS

3.14.5 Solución

Ver test 2.11

3.15 Test 2.34 -Part A: Unicast Neighbor Solicitation with the same SLLA

3.15.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado STALE.

3.15.2 Procedimiento

1. TN1 envía un Echo Request (A).
3. TN1 envía un NA solicitado (B).
5. TN1 envía un Echo Request (A).
7. Espera (REACHABLE_TIME * MAX_RANDOM_FACTOR) segundos.
9. TN1 envía un NS (C).
10. TN1 envía un Echo Request al NUT.

3.15.3 Resultados esperados

2. El NUT debe crear un NCE para TN1 y setear su estado a. El NUT debe enviar un multicast NS a TN1.
4. Tras recibir el NA de TN1, el NUT debe enviar el Echo Reply encolado a TN1. El NUT debe actualizar el NCE de TN1 a estado REACHABLE y actualizar el Link-layer address de TN1.
6. Ya que el NUT está en estado REACHABLE, tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.
8. El NUT debe actualizar el NCE de TN1 a estado STALE.
11. El NUT NO debe actualizar el NCE de TN1 y mantenerse en estado STALE. El NUT debe responder al NS enviando un NA. Tras responder al NS, el NUT debe responder al Echo Request enviando un Echo Reply y setear el NCE de TN1 a estado DELAY. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1.

3.15.4 Resultado obtenido

Couldn't observe NS.

Falla paso 2 ya que no se recibe el NS.

3.15.5 Solución

Ver Test 2.15

3.16 Test 2.35 - Part B: Unicast Neighbor Solicitation with a different SLLA

3.16.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado STALE.

3.16.2 Procedimiento

1. TN1 envía un Echo Request (A).
3. TN1 envía un NA solicitado (B).
5. TN1 envía un Echo Request (A).
7. Espera (REACHABLE_TIME * MAX_RANDOM_FACTOR) segundos.
9. TN1 envía un NS (C) con un SLLA diferente.
10. TN1 envía un Echo Request al NUT.

3.16.3 Resultados esperados

2. El NUT debe crear un NCE para TN1 y setear su estado a INCOMPLETE. El NUT debe enviar un multicast NS a TN1.
4. Tras recibir el NA de TN1, el NUT debe enviar el Echo Reply encolado a TN1. El NUT debe actualizar el estado de TN1 a REACHABLE y su Link-layer address.
6. Como el NUT está en estado REACHABLE, tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.
8. El NUT debe actualizar el NCE de TN1 a estado STALE.
11. El NUT debe actualizar el Link-layer address de TN1 a la nueva dirección que viene en el NS C. El NUT NO debe actualizar el NCE de TN1 y mantenerse en estado STALE. El NUT debe responder al NS enviando un NA al nuevo Link-Layer address de TN1. Tras responder al NS, el NUT debe responder al Echo Request enviando un Echo Reply y setear el estado de TN1 a DELAY. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1 usando como Target el nuevo Link-layer address.

3.16.4 Resultado obtenido

Observed unexpected packet

Se esperaba un NS, pero se recibió un Echo Reply

3.16.5 Solución

Ver Test 2.11

3.17 Test 2.37 - Part D: Multicast Neighbor Solicitation with a different SLLA

3.17.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado STALE.

3.17.2 Procedimiento

1. TN1 envía un Echo Request (A).
3. TN1 envía un NA solicitado (B).
5. TN1 envía un Echo Request (A).
7. Espera (REACHABLE_TIME * MAX_RANDOM_FACTOR) segundos.
9. TN1 envía un NS (C) con un SLLA diferente.
10. TN1 envía un Echo Request al NUT.

3.17.3 Resultados esperados

2. El NUT debe crear un NCE para TN1 y setear su estado a INCOMPLETE. El NUT debe enviar un multicast NS a TN1.
4. Tras recibir el NA de TN1, el NUT debe enviar el Echo Reply encolado a TN1. El NUT debe actualizar el estado de TN1 a REACHABLE y su Link-layer address.
6. Como el NUT está en estado REACHABLE, tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.
8. El NUT debe actualizar el NCE de TN1 a estado STALE.
11. El NUT debe actualizar el Link-layer address de TN1 a la nueva dirección que viene en el NS C. El NUT NO debe actualizar el NCE de TN1 y mantenerse en estado STALE. El NUT debe responder al NS enviando un NA al nuevo Link-Layer address de TN1. Tras responder al NS, el NUT debe responder al Echo Request enviando un Echo Reply y setear el estado de TN1 a DELAY. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1 usando como Target el nuevo Link-layer address.

3.17.4 Resultado obtenido

Couldn't observe NS

Se esperaba un NS, pero se recibió un Echo Reply

3.17.5 Solución

Ver Test 2.11

3.18 Test 2.38 - Part A: Unicast Neighbor Solicitation with the same SLLA

3.18.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado PROBE.

3.18.2 Procedimiento

1. TN1 envía un Echo Request (A).
2. TN1 envía un NA solicitado (B) al recibir cualquier NS.
4. Espera (DELAY_FIRST_PROBE_TIME) segundos.
6. TN1 envía un NS (C).
7. TN1 envía un Echo Request al NUT.

3.18.3 Resultados esperados

3. El NUT debe actualizar el NCE de TN1 a estado STALE. Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply y pasar a estado DELAY.
5. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe pasar a estado PROBE y enviar un NS a TN1.
8. El NUT NO debe actualizar el estado de TN1 tras enviar el NA y el Echo Reply encolados, y mantenerse en estado PROBE. El NUT debe retransmitir el NS a TN1.

3.18.4 Resultado obtenido

Couldn't observe NS

3.18.5 Solución

Ver test 2.41

3.19 Test 2.39 - Part B: Unicast Neighbor Solicitation with a different SLLA

3.19.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado STALE.

3.19.2 Procedimiento

1. TN1 envía un Echo Request (A).
2. TN1 envía un NA solicitado (B) al recibir cualquier NS.
4. Espera (DELAY_FIRST_PROBE_TIME) segundos.
6. TN1 envía un NS (C) con una opción SLLA diferente.
7. TN1 envía un Echo Request al NUT.

3.19.3 Resultados esperados

3. El NUT debe actualizar el NCE de TN1 a estado STALE. Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply y pasar a estado DELAY.
5. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe pasar a estado PROBE y enviar un NS a TN1.
8. El NUT debe actualizar el Link-layer address con el recibido en el NS C y actualizar el estado de TN1 a STALE. El NUT debe responder al NS enviando un NA usando el Link-layer address de TN1. Tras responder al NS, el NUT debe responder al Echo Request enviando un Echo Reply y setear el estado de TN1 a DELAY. Pasado, DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1 usando como Target el nuevo Link-layer address de TN1.

3.19.4 Resultado obtenido

Couldn't observe NS

3.19.5 Solución

Ver test 2.41

3.20 Tests 2.40 - Part C: Multicast Neighbor Solicitation with the same SLLA

3.20.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado STALE.

3.20.2 Procedimiento

1. TN1 envía un Echo Request (A).
2. TN1 envía un NA (B) al recibir cualquier NS.
4. Espera (DELAY_FIRST_PROBE_TIME) segundos.
6. TN1 envía un NS (D).
7. TN1 envía un Echo Request al NUT.

3.20.3 Resultados esperados

3. El NUT debe actualizar el NCE de TN1 a estado STALE. Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply y pasar a estado DELAY.
5. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe pasar a estado PROBE y enviar un NS a TN1.
8. El NUT NO debe actualizar el estado de TN1 tras enviar el NA y el Echo Reply encolados, y mantenerse en estado PROBE. El NUT debe retransmitir el NS a TN1.

3.20.4 Resultado obtenido

Couldn't observe NS

3.20.5 Solución

Ver test 2.41

3.21 Test 2.41 - Part D: Multicast Neighbor Solicitation with a different SLLA

3.21.1 Propósito

Verificar que un nodo actualiza apropiadamente su Neighbor Cache cuando recibe NS's y el NCE para ese vecino (neighbor) está en estado STALE.

3.21.2 Procedimiento

1. TN1 envía un Echo Request (A).
2. TN1 envía un NA (B) al recibir cualquier NS.
4. Espera (DELAY_FIRST_PROBE_TIME) segundos.
6. TN1 envía un NS (D) con una opción SLLA diferente.
7. TN1 envía un Echo Request al NUT.

3.21.3 Resultados esperados

3. El NUT debe actualizar el NCE de TN1 a estado STALE. Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply y pasar a estado DELAY.
5. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe pasar a estado PROBE y enviar un NS a TN1.
8. El NUT debe actualizar el Link-layer address con el recibido en el NS C y actualizar el estado de TN1 a STALE. El NUT debe responder al NS enviando un NA usando el Link-layer address de TN1. Tras responder al NS, el NUT debe responder al Echo Request enviando un Echo Reply y setear el estado de TN1 a DELAY. Pasado, DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1 usando como Target el nuevo Link-layer address de TN1.

3.21.4 Resultado obtenido

Couldn't observe NS

Falló el paso 5.

3.21.5 Solución

Estos tests fallan debido a que no se pasa correctamente al estado PROBE cuando se recibe el primer NA en el paso 2.

Esto causa que no se envíen los NS's que se piden en el paso 5, y que por ende falle el test.

Para solucionar esto se modificó la clase

`ip4jvm.net.applications.NeighborDiscovery`

Método:

```
rcvICMPv6NeighborAdvertisement(ICMPv6Header, IPv6ProtocolHeader,
IPv6NetParameters, String, NetConfigs)
```

Para que funcionara este test, se realizaron una serie de cambios en este método para que realice correctamente el cambio de estado de:

INCOMPLETE -> DELAY -> PROBE

3.22 Tests 2.45 al 2.51 (Invalid Neighbor Advertisement Handling)

3.22.1 Propósito

Verificar que un nodo actúa apropiadamente al recibir un Neighbor Advertisement (NA) inválido.

3.22.2 Procedimiento

1. TN1 envía un Echo Request al NUT.
3. TN1 envía un NA (A) inválido:
   - Test 2.45 - con la Flag Solicited = 1.
   - Test 2.46 - con el campo Hop Limit = 254.
   - Test 2.47 - con el campo Checksum con un valor inválido.
   - Test 2.48 - con el campo ICMP Code != 0.
   - Test 2.49 - con el campo ICMP Length < 24.
   - Test 2.50 - con el Target address = multicast address.
   - Test 2.51 - con el campo Option Length = 0.

3.22.3 Resultados esperados

2. El NUT envía un NS al solicited-node multicast address de TN1's.
4. El NUT debe ignorar el NA enviado por TN1 y continuar enviando NS’s al solicited-node multicast address de TN1's.

3.22.4 Resultado obtenido

Couldn't observe NS

3.22.5 Solución

Todos estos tests se resuelven con lo expuesto en el test 2.11.

3.23 Tests 2.52 al 2.59 (Neighbor Advertisement Processing, No NCE – excepto 2.53 y 2.55)

3.23.1 Propósito

Verificar que un nodo descarta silenciosamente un Neighbor Advertisement (NA) si el target no tiene un Neighbor Cache entry (NCE).

3.23.2 Procedimiento

1. TN1 envía un A. Para cada test se envía un NA diferente:
Test 2.52 - Part A: El NA tiene los campos S = 0, O = 0, y opción TLLA
Test 2.54 - Part C: El NA tiene los campos S = 1, O = 0, y opción TLLA
Test 2.56 - Part E: El NA tiene los campos S = 0, O = 0, y NO tiene opción TLLA
Test 2.57 - Part F: El NA tiene los campos S = 0, O = 1, y NO tiene opción TLLA
Test 2.58 - Part G: El NA tiene los campos S = 1, O = 0, y NO tiene opción TLLA

3. TN1 envía un Echo Request al NUT.

3.23.3 Resultados esperados

2. Para cada parte, tras recibir el NA de TN1, el NUT NO debe enviar ningún paquete y no se debe crear un NCE para TN1. Tras recibir el Echo Request de TN1, el NUT debe crear un NCE para TN1 setear su estado a INCOMPLETE. El NUT debe enviar un multicast NS a TN1.

3.23.4 Resultado obtenido

En cada parte se obtuvo el mismo error:

Couldn't observe NS

3.23.5 Solución

Todos estos tests se resuelven con lo expuesto en el test 2.11.

3.24 Test 2.60 al 2.64 (Neighbor Advertisement Processing, NCE State INCOMPLETE)

3.24.1 Propósito

Verificar que un nodo actualiza apropiadamente el Neighbor Cache del estado INCOMPLETE al recibir un NA.

3.24.2 Procedimiento

1. TN1 envía un Echo Request A.
2. TN1 envía un NA. Para cada uno de los tests, el NA es diferente.
   - Test 2.60: El NA tiene los campos S = 1 y O = 1
   - Test 2.61: El NA tiene los campos S = 1 y O = 0
   - Test 2.62: El NA tiene los campos S = 0 y O = 1
   - Test 2.63: El NA tiene los campos S = 0 y O = 0
   - Test 2.64: El NA no tiene opción TTLA
5. TN1 envía un Echo Request (Solo aplica para Tests 2.60 y 2.61).

3.24.3 Resultados esperados

2. Tras recibir un Echo Request de TN1, el NUT debe crear un NCE para TN1 y setear su estado a INCOMPLETE. El NUT envía un NS a TN1.
4a. (Tests 2.60 y 2.61): Al recibir el NA de TN1, el NUT debe enviar el Echo Reply encolado a TN1 y actualizar el estado del NCE de TN1 a estado REACHABLE (Tests 2.60 y 2.61) ó a estado STALE (Tests 2.62 y 2.63). Además se actualiza el Link-layer address de TN1 (excepto en el Test 2.64). En el caso del test 2.64 se ignora el NA y el estado se mantiene en INCOMPLETE.
4b.
6. (Solo aplica para Tests 2.60 y 2.61): Ya que el NUT queda en estado REACHABLE, tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.

3.24.4 Resultado obtenido

En los cinco casos el error obtenido fue:

Couldn’t observe NS

3.24.5 Solución

Los tests 2.60, 2.61, 2.64 y 2.65 quedan resueltos gracias al test 2.11.

Para el 2.62 y 2.63 se agrega el siguiente código a la clase

```
ip4jvm.net.applications.NeighborDiscovery
```

Método:

```
rcvICMPv6NeighborAdvertisement(ICMPv6Header, IPv6ProtocolHeader,
IPv6NetParameters, String, NetConfigs)
```

Mas precisamente agregar el siguiente código cuando se chequea la siguiente condición

```
a!=null && entryState == NeighborCache.STATE_INCOMPLETE && !solicited

addNeighborHost(jth, mac, icmpNA.targetAddress, icmpNA.routerFlag,
                NeighborCache.STATE_STALE, false);
setState(icmpNA.targetAddress, NeighborCache.STATE_DELAY,
         (IPv6NetConfig) netConfig.getFirst());
```

Esto realiza el cambio de estado como se especifica en el punto 4a.

3.25 Tests 2.65 al 2.82 (Neighbor Advertisement Processing, NCE State REACHABLE – excepto 2.67, 2.71, 2.75, 2.78 y 2.82)

3.25.1 Propósito

Verificar que un nodo actualiza apropiadamente el Neighbor Cache del estado REACHABLE al recibir un NA.

3.25.2 Procedimiento

1. TN1 envía un Echo Request A.
3. TN1 envía un NA al NUT.
5. TN1 envía un NA. Las flags Solicited y Override; así como la opción TTLA se setean como se describe a continuación.

Test	Parte	Destination	Solicited	Override	TTLA	Nuevo Estado	Actualizar LLA
65	A	Unicast	Clear	Clear	None	REACHABLE	no
66	B	Unicast	Clear	Set	None	REACHABLE	no
67	C	Unicast	Set	Clear	None	REACHABLE	no
68	D	Unicast	Set	Set	None	REACHABLE	no
69	E	Unicast	Clear	Clear	Same	REACHABLE	no
70	F	Unicast	Clear	Set	Same	REACHABLE	no
71	G	Unicast	Set	Clear	Same	REACHABLE	no
72	H	Unicast	Set	Set	Same	REACHABLE	no
73	I	Unicast	Clear	Clear	Different	STALE	no
74	J	Unicast	Clear	Set	Different	STALE	yes
75	K	Unicast	Set	Clear	Different	STALE	no
76	L	Unicast	Set	Set	Different	REACHABLE	yes
77	M	Multicast	Clear	Clear	Same	REACHABLE	no
78	N	Multicast	Clear	Set	Same	REACHABLE	no
79	O	Multicast	Clear	Clear	Different	STALE	no
80	P	Multicast	Clear	Set	Different	STALE	yes
81	Q	Multicast	Clear	Clear	None	REACHABLE	no
82	R	Multicast	Clear	Set	None	REACHABLE	no

6. TN1 envía un Echo Request.

3.25.3 Resultados esperados

2. El NUT debe crear un NCE para TN1 y setear el estado a INCOMPLETE. El NUT debe enviar un NS a TN1.
4. Ya que el NUT está en estado REACHABLE, tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el

NUT no debe enviar un NS a TN1.
7. El NUT debe actualizar el estado del NCE y el LLA de TN1 de acuerdo a lo siguiente.

Partes A-H, L-N y Q-R a REACHABLE
Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.
(Solo Parte L) El Echo Reply debe ser enviado al nuevo Link-layer destination address de TN1.

Partes I-K, y O-P a STALE
Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1.
(Solo partes J y P) El Echo Reply y el NS deben ser enviados al nuevo Link-layer destination address de TN1.

3.25.4 Resultado obtenido

Tests A, D, E, H, I, J, L, O, P - Couldn't observe NS
Tests B, F, M, Q - Observed unexpected packet

3.25.5 Solución

Todos los tests quedan resueltos gracias al test 2.11.

3.26 Tests 2.83 al 2.100 (Neighbor Advertisement Processing, NCE State STALE – excepto 86, 90, 94, 98)

3.26.1 Propósito

Verificar que un nodo actualiza apropiadamente el Neighbor Cache del estado STALE al recibir un NA.

3.26.2 Procedimiento

1. TN1 envía un Echo Request A.
3. TN1 envía un NA al NUT.
5. Espera (REACHABLE_TIME * MAX_RANDOM_FACTOR) segundos.
7. TN1 envía un NA. Las flags Solicited y Override; así como la opción TTLA se setean como se describe a continuación.

Test	Parte	Destination	Solicited	Override	TTLA	Nuevo Estado	Actualizar LLA
83	A	Unicast	Clear	Clear	None	STALE	no
84	B	Unicast	Clear	Set	None	STALE	no
85	C	Unicast	Set	Clear	None	REACHABLE	no
86	D	Unicast	Set	Set	None	REACHABLE	no
87	E	Unicast	Clear	Clear	Same	STALE	no
88	F	Unicast	Clear	Set	Same	STALE	no
89	G	Unicast	Set	Clear	Same	REACHABLE	no
90	H	Unicast	Set	Set	Same	REACHABLE	no
91	I	Unicast	Clear	Clear	Different	STALE	no
92	J	Unicast	Clear	Set	Different	STALE	yes
93	K	Unicast	Set	Clear	Different	STALE	no
94	L	Unicast	Set	Set	Different	REACHABLE	yes
95	M	Multicast	Clear	Clear	Same	STALE	no
96	N	Multicast	Clear	Set	Same	STALE	no
97	O	Multicast	Clear	Clear	Different	STALE	no
98	P	Multicast	Clear	Set	Different	STALE	yes
99	Q	Multicast	Clear	Clear	None	STALE	no
100	R	Multicast	Clear	Set	None	STALE	no

8. TN1 envía un Echo Request.

3.26.3 Resultados esperados

2. El NUT debe crear un NCE para TN1 y setear el estado a INCOMPLETE. El NUT debe enviar un NS a TN1.
4. Ya que el NUT está en estado REACHABLE, tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT no debe enviar un NS a TN1.

6. El NUT debe cambiar el estado de TN1 a STALE.
9. El NUT debe actualizar el estado del NCE y el LLA de TN1 de acuerdo a lo siguiente.

Partes C, D, G, H y L a REACHABLE
Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT NO debe enviar un NS a TN1.
(Solo Parte L) El Echo Reply debe ser enviado al nuevo Link-layer destination address de TN1.

Partes A, B, E, F, I-K, y M-R a STALE
Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1.
(Solo partes J y P) El Echo Reply y el NS deben ser enviados al nuevo Link-layer destination address de TN1.

3.26.4 Resultado obtenido

Tests A, B, E, F, I, J, M, N, Q - Couldn't observe NS

Tests C, G, K, O, R - Observed unexpected packet

3.26.5 Solución

Todos estos tests son resueltos con lo descrito en los test 2.11 y 2.15.

3.27 Tests 2.101 al 2.118 (Neighbor Advertisement Processsing, NCE State PROBE)

3.27.1 Propósito

Verificar que un nodo actualiza apropiadamente el Neighbor Cache del estado PROBE al recibir un NA.

3.27.2 Procedimiento

1. TN1 envía un Echo Request A al NUT.
2. TN1 envía un NA al NUT.
4. Esperar (DELAY_FIRST_PROBE_TIME) segundos.
6. TN1 envía un NA A. Los campos Solicited y Override se setean de acuerdo a la parte A de la tabla descrita más abajo. De la misma forma, la dirección en la opción TLLA se describe en la tabla.

Test	Parte	Destination	Solicited	Override	TTLA	Nuevo Estado	Actualizar LLA
101	A	Unicast	Clear	Clear	None	PROBE	no
102	B	Unicast	Clear	Set	None	PROBE	no
103	C	Unicast	Set	Clear	None	REACHABLE	no
104	D	Unicast	Set	Set	None	REACHABLE	no
105	E	Unicast	Clear	Clear	Same	PROBE	no
106	F	Unicast	Clear	Set	Same	PROBE	no
107	G	Unicast	Set	Clear	Same	REACHABLE	no
108	H	Unicast	Set	Set	Same	REACHABLE	no
109	I	Unicast	Clear	Clear	Different	PROBE	no
110	J	Unicast	Clear	Set	Different	STALE	yes
111	K	Unicast	Set	Clear	Different	PROBE	no
112	L	Unicast	Set	Set	Different	REACHABLE	yes
113	M	Multicast	Clear	Clear	Same	PROBE	no
114	N	Multicast	Clear	Set	Same	PROBE	no
115	O	Multicast	Clear	Clear	Different	PROBE	no
116	P	Multicast	Clear	Set	Different	STALE	yes
117	Q	Multicast	Clear	Clear	None	PROBE	no
118	R	Multicast	Clear	Set	None	PROBE	no

7. (Esta parte no aplica para A, B, E, F, I, K, M, N, O, Q y R) TN1 envía un Echo Request.

3.27.3 Resultados esperados

3. El NUT debe cambiar el estado del NCE de TN1 a STALE. Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply y pasar al estado DELAY.
5. El NUT debe cambiar el estado del NCE de TN1 a PROBE enviando un NS a TN1.
8. El NUT debe actualizar el estado del NCE y el LLA de TN1 de acuerdo a lo siguiente.

Partes C, D, G, H y L a REACHABLE
Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT no debe enviar un NS a TN1.

Parte L
El Echo Reply debe ser enviado al nuevo Link-layer address de TN1.

Partes J y P a STALE
Tras recibir el Echo Request de TN1, el NUT debe enviar un Echo Reply. Pasado DELAY_FIRST_PROBE_TIME, el NUT debe enviar un NS a TN1. El Echo Reply debe ser enviado al nuevo Link-layer address de TN1. El NS debe usar como Target, el nuevo Link-layer address.

Partes A, B, E, F, I, K, M, N, O, Q y R a PROBE
El NUT debe enviar un NS a TN1.

3.27.4 Resultado obtenido

En todos estos casos se obtuvo el siguiente error:

Couldn't observe NS

3.27.5 Solución

Se modificó la forma que se maneja la recepción de los NA cuando el estado del NCE es PROBE. Se siguió los lineamientos planteados en el punto 6 del apartado 1.2.27.2

La clase modificada fue:

```
ip4jvm.net.applications.NeighborDiscovery
```

Método:

```
rcvICMPv6NeighborAdvertisement(ICMPv6Header, IPv6ProtocolHeader,
IPv6NetParameters, String, NetConfigs)
```

No se profundiza en cada cambio particular ya que fueron necesarios varios cambios consecutivos para hacer funcionar los 18 tests.

3.28 Tests 2.119 al 2.126 (Neighbor Advertisement Processing, R-bit Change (Hosts Only))

3.28.1 Propósito

Verificar que un nodo toma las acciones apropiadas cuando un vecino que es un Router comienza a transmitir los NA con la flag Router clear (en 0).

3.28.2 Procedimiento

1. TR1 envía un RA sin opción SLLA.
2. TN1 envía un Echo Request A.
4. TR1 responde a NS's del HUT con un NA con las flags Router, Solicited y Override seteadas de acuerdo a lo siguiente:

Test 2.119: El NA tiene los campos S = 1 y O = 1 y NO tiene opción SLLA.
Test 2.120: El NA tiene los campos S = 0 y O = 0 y NO tiene opción SLLA.
Test 2.121: El NA tiene los campos S = 0 y O = 1 y NO tiene opción SLLA.
Test 2.122: El NA tiene los campos S = 1 y O = 0 y NO tiene opción SLLA.
Test 2.123: El NA tiene los campos S = 1 y O = 1 y tiene opción SLLA.
Test 2.124: El NA tiene los campos S = 0 y O = 0 y tiene opción SLLA.
Test 2.125: El NA tiene los campos S = 0 y O = 1 y tiene opción SLLA.
Test 2.126: El NA tiene los campos S = 1 y O = 0 y tiene opción SLLA.

6. TR1 envía un NA. Para cada parte del test, se envía un NA diferente.
7. TN1 envía un Echo Request A.

3.28.3 Resultados esperados

3. El HUT debe solicitar TR1 enviando NS's con Target = Link-local Address de TR1.
5. El HUT debe enviar un Echo Reply usando a TR1 como "first hop".
8. El HUT NO debe enviar un Echo Reply usando a TR1 como "first hop" en respuesta al paquete A del paso 7 y el HUT NO debe enviar NS's con Target = Link-local Address de TR1.

3.28.4 Resultado obtenido

Couldn't observe Echo Reply

Falla el paso número 5.

3.28.5 Solución

El problema en estos tests es que cuando se recibe un NA con la flag isRouter = 0, se debe responder enviando el Echo Reply que está en espera.

Para esto, la clase modificada fue:

`ip4jvm.net.applications.NeighborDiscovery`

Método:

```
rcvICMPv6NeighborAdvertisement(ICMPv6Header, IPv6ProtocolHeader,
IPv6NetParameters, String, NetConfigs)
```

Se agrega el siguiente código al final del método:

```
if (icmpNA.routerFlag && mac != null) {
      LinkedList<Inet6Address> l =
            destinationCache.getRouterNEIGH(icmpNA.targetAddress);
      for (Inet6Address inet6Address : l) {
            stack.activateJobs(2, inet6Address, mac, jth);
      }
}
```

Esto envía los Echo Reply pendientes para TR1 (`icmpNA.targetAddress`)

3.29 Tests 2.128 al 2.135 (Router Solicitations)

3.29.1 Propósito

Verificar que un host envía los RS válidos en el momento correcto.

3.29.2 Procedimiento

N/A

3.29.3 Resultados esperados

N/A

3.29.4 Resultado obtenido

Initialization fail.

3.29.5 Solución

Estos tests fallaron por problemas en la configuración de FreeBSD. Se enviaban paquetes ajenos a los tests de TAHI, lo que hacía que los tests fallaran.

En caso de ocurrir revisar que en /etc/rc.conf se encuentre la siguiente configuración:

```
ipv6_enable="NO"
```

3.30 Tests 2.136 y 2.137 (Host Ignores Router Solicitations)

3.30.1 Propósito

Verificar que un host ignora Los RS y no actualiza el Neighbor Cache.

3.30.2 Procedimiento

1. TN1 envía un RS A. El Destination Address = All-Router multicast Address (Test 136) y Link-local address del HUT (Test 137).
2. Esperar (RETRANS_TIMER * MAX_*CAST_SOLICIT). (3 segundos)
3. TN1 envía un Echo Request al HUT.
4. Esperar 2 segundos.

3.30.3 Resultados esperados

5. En ambas partes, el HUT debe enviar un multicast NS para TN1, lo cual indica que el HUT NO proceso el RS de TN1.

3.30.4 Resultado obtenido

En ambos casos se obtuvo: *Couldn't observe NS*

3.30.5 Solución

Ver Test 2.11.

3.31 Test 2.138 - Default Router Switch

3.31.1 Propósito

Verificar que si un host mantiene al menos dos routers en la Default Router List va a cambiar (switch) entre routers en caso que uno de ellos falle.

3.31.2 Procedimiento

1. TR1 envía un Router Advertisement.
2. TN2 envía un Echo Request A.
4. TR2 envía otro Router Advertisement diferente al de la parte 1.
5. TN2 envía el paquete A cada 3 segundos por 30 segundos.
7. Cuando expira el Reachable Time, y el HUT solicita a TR1, no se envían los NA desde TR1.

3.31.3 Resultados esperados

3. El HUT debe enviar un NS con Target Address = Link-local address de TR1. El HUT debe enviar un Echo Reply a TN2 a través de TR1 en respuesta al paquete A.
6. El HUT debe enviar Echo Replies al Link-local address de TR1 hasta que expira el Reachable Time. Cuando expira el Reachable Time, el HUT debe enviar 3 NS's al Link-local address de TR1.
8. El HUT selecciona a TR2 de la Default Router list. El HUT envía NS's al Link-local address de TR2. Tras enviar los paquetes a TR2, el HUT pasa a TR2 a estado PROBE como efecto lateral.

3.31.4 Resultado obtenido

Couldn't observe Echo Reply

3.31.5 Solución

Modificar el método: StateTimeout
Se agregaron las siguientes líneas de código

```
if (defaultRouterList.hasAddress(jth, addr) &&
    defaultRouterList.getNumberOfRouters() > 1){
      destinationCache.invalidateRouter(jth, addr);
      defaultRouterList.deleteItem(jth, addr);
}
```

En este caso si tenemos más de un router dentro de la Default Router List, eliminamos este router para que se seleccione otro.

3.32 Tests 2.139 al 2.144 (Router Advertisement Processing, Validity)

3.32.1 Propósito

Verificar que un host descarta apropiadamente un RA inválido.

3.32.2 Procedimiento

N/A

3.32.3 Resultados esperados

N/A

3.32.4 Resultado obtenido

N/A

3.32.5 Solución

Estos tests fallaron por problemas en la configuración de FreeBSD. Se enviaban paquetes ajenos a los tests de TAHI, lo que hacía que los tests fallaran.

En caso de ocurrir revisar que en /etc/rc.conf se encuentre la siguiente configuración:

```
ipv6_enable="NO"
```

3.33 Tests 2.145 y 2.146 (Router Advertisement Processing, Cur Hop Limit)

3.33.1 Propósito

Verificar que un nodo procesa apropiadamente el campo Cur Hop Limit de un RA.

3.33.2 Procedimiento

1. TN1 envía un Echo Request al NUT.
3. Si el NUT es un host, TR1 envía un RA con Cur Hop Limit = 0 (Test 145) y Cur Hop Limit = 15 (Test 146). Si el NUT es un router, configurar el Cur Hop Limit a 0 (Test 145) y a 15 (Test 146).
4. TN1 envía un Echo Request al NUT.

3.33.3 Resultados esperados

2. El NUT debe responder al Echo Request de TN1. Observar el Hop Limit en el Echo Reply que envía el NUT.
3. Si el NUT es un router, el NUT debe enviar un RA con Cur Hop Limit en 0 (Test 145) y en 15 (Test 146).
5. El NUT debe responder al Echo Request de TN1. El Cur Hop Limit en el Echo Reply debe ser 0 (Test 145) y 15 (Test 146).

3.33.4 Resultado obtenido

Couldn't observe NS

3.33.5 Solución

El test 2.145 queda resuelto con lo expuesto en el test 2.11.

Para el test 2.146 además hay que modificar la forma en que se envía el Echo Reply para que coincida con el enviado por el Echo Request (en este caso 15).

3.34 Tests 2.150 y 2.151 (Router Advertisement Processing, Reachable Time)

3.34.1 Propósito

Verificar que un nodo actualiza el BaseReachableTime y recalcula el ReachableTime al recibir un Router Advertisement o una configuración con un Reachable Time especificado.

3.34.2 Procedimiento

Part A: RA Processing – Reachable Time (Host Only)

1. TR1 envía un RA con un Router Lifetime de 0 segundos y un Reachable Time de 10 segundos.
2. TN1 envía un Echo Request al HUT. TN1 debe responder a cualquier NS del HUT.
4. Repetir el Paso 2 cada segundo durante 40 segundos.
6. TR1 envía un RA con un Reachable Time de 40 segundos.
7. Repetir el Paso 2 cada segundo durante 140 segundos.

Part B: Reachable Time Configuration (Routers Only)

1. Configurar el RUT para enviar un RA con un Router Lifetime de 0 segundos y un Reachable Time de 10 segundos.
2. TN1 envía un Echo Request al HUT. TN1 debe responder a cualquier NS del RUT.
4. Repetir el Paso 2 cada segundo durante 40 segundos.

3.34.3 Resultados esperados

Parte A
3. El HUT debe solicitar el Link-local address de TN1 y enviar un Echo Reply.
5. El HUT debe enviar un NS con Target Address = Link-local address de TN1 en un intervalo de entre 10 y 20 segundos. [Reachable Time (5 a 15 seg.) + DELAY_FIRST_PROBE_TIME (5 seg.)].
8. El HUT debe enviar un NS en un intervalo de entre 25 y 65 segundos. [Reachable Time (20 a 60 seg.) + DELAY_FIRST_PROBE_TIME (5 seg.)].

Parte B
3. El RUT debe solicitar el Link-local address de TN1 y enviar un Echo Reply.
5. El RUT debe enviar un NS con Target Address = Link-local address de TN1 en un intervalo de entre 10 y 20 segundos. [Reachable Time (5 a 15 seg.) + DELAY_FIRST_PROBE_TIME (5 seg.)].

3.34.4 Resultado obtenido

Couldn't observe NS

3.34.5 Solución

Estos tests quedan resueltos con lo expuesto en el test 2.15.

3.35 Tests 2.152 al 2.162 (Router Advertisement Processing, Neighbor Cache – excepto 152, 154, 156 y 160)

3.35.1 Propósito

Verificar que un host actualiza apropiadamente el Neighbor Cache al recibir un Router Advertisement (RA).

3.35.2 Procedimiento

Se envía el mismo RA, con TN1 estando en diferentes estados:
153. Part B: RA con NCE INCOMPLETE
155. Part D: RA con misma opción SLLA, NCE REACHABLE
157. Part F: RA con diferente opción SLLA, NCE PROBE
158. Part G: RA con misma opción SLLA, NCE PROBE
159. Part I: RA con diferente opción SLLA, NCE STALE
161. Part K: RA sin opción SLLA, NCE STALE
162. Part K: RA sin opción SLLA, NCE STALE.

Por más detalles ver 177-178 del documento de Core Conformance.

3.35.3 Resultados esperados

Ver páginas 178-181 del documento de Core Conformance.

3.35.4 Resultado obtenido

Couldn't observe NS

3.35.5 Solución

Estos tests quedan resueltos con lo expuesto en el test 2.15.

3.36 Test 2.163 - Part A: RA without Source Link-layer option

3.36.1 Propósito

Verificar que un nodo actualiza apropiadamente la flag IsRouter en el Neighbor Cache el recibir un RA.

3.36.2 Procedimiento

1. TR1 envía un Echo Request al HUT.
2. TR1 responde a cualquier NS con un NA (R=0, S=1, O=1).
4. TR1 envía un RA sin opción SLLA al NUT.
5. Esperar a que el HUT realice Duplicate Address Detection.
6. TN2 envía un Echo Request A al HUT tal que el Next Hop = TR1.

3.36.3 Resultados esperados

3. El HUT debe enviar un Echo Reply al Link-local address de TR1 y actualiza su NCE a estado REACHABLE. El HUT setea la flag isRouter a false.
7. El HUT setea la flag isRouter a true y envía un Echo Reply al Off-Link address de TN2 con Next Hop = TR1.

3.36.4 Resultado obtenido

Couldn't observe NS

3.36.5 Solución

Este test queda resuelto con lo expuesto en el test 2.11.

3.37 Test 2.165 - Part C: RA with different Source Link-layer option as cached

3.37.1 Propósito

Verificar que un nodo actualiza apropiadamente la flag IsRouter en el Neighbor Cache el recibir un RA.

3.37.2 Procedimiento

1. TR1 envía un Echo Request al HUT.
2. TR1 responde a cualquier NS con un NA (R=0, S=1, O=1).
4. TR1 envía un RA con una opción SLLA diferente al NUT.
5. Esperar a que el HUT realice Duplicate Address Detection.
6. TN2 envía un Echo Request A al HUT tal que el Next Hop = TR1.

3.37.3 Resultados esperados

3. El HUT debe enviar un Echo Reply al Link-local address de TR1 y actualiza su NCE a estado REACHABLE. El HUT setea la flag isRouter a false.
7. El HUT setea la flag isRouter a true y envía un Echo Reply al Off-Link address de TN2 con Next Hop = TR1.

3.37.4 Resultado obtenido

Couldn’t observe NS

3.37.5 Solución

Este test queda resuelto con lo expuesto en el test 2.11.

3.38 Tests 2.169 al 2.236 – Tests Redirect

3.38.1 Propósito

Estos tests cubren una variedad de tests que refieren al proceso de Redirect.

3.38.2 Procedimiento

N/A

3.38.3 Resultados esperados

N/A

3.38.4 Resultado obtenido

N/A

3.38.5 Solución

La totalidad de los tests de Redirect fallaron ya que quedaban entradas dentro del Neighbor Cache que provocaba que luego de que fallara un test; todos los restantes tests fallaran también. La solución (muy básica) fue que tras realizar el clean-up del test TAHI se limpie el Destination Cache.

Clase:

```
ip4jvm.net.applications.NeighborDiscovery
```

Método:

```
rcvICMPv6RouterAdvertisement(ICMPv6Header, IPv6ProtocolHeader,
IPv6NetParameters, String)
```

Al chequear la condición `icmpRA.getRouterLifeTime()==0`

Se limpia el Destination Cache, de la siguiente manera:

```
destinationCache = new DestinationCache();
```

Queda como pendiente mejorar este método de limpieza del Neighbor Cache.

4 Section III - RFC 4862 - IPv6 Stateless Address Autoconfiguration

Los siguientes tests cubren la especificación del IPv6 Stateless Address Autoconfiguration, Request For Comments 4862. Estos tests verifican el proceso para generar el Link-local address, el proceso para generar Site-local y Global addresses a través de Stateless Address Autoconfiguration, y el proceso de Duplicate Address Detection. Además verifican que un host procesa correctamente un Router Advertisement y asigna correctamente los “lifetimes”.

Inicialmente todos los tests de la sección fallaron, sin embargo:

Los tests 3.2, 3.4, 3.6-3.13, 3.16-3.26, 3.31-3.35, 3.38, 3.40, 3.42

en realidad no fallan, sino que se estaba ejecutando los tests en sentido incorrecto.

Usualmente un test TAHI se ejecuta con el nodo ip4jvm (stack) ya inicializado. Sin embargo el Address Autoconfiguration and Duplicate Address Detection es un proceso que se da al iniciar del nodo, teniendo el nodo “APAGADO”, debemos inicializar el test del lado del TN y luego iniciar el proceso en el NUT. Con esta sencilla consideración, pasamos de 45 tests fallidos a 17.

1	FAIL	13	PASS	25	PASS	37	FAIL
2	PASS	14	FAIL	26	PASS	38	PASS
3	FAIL	15	FAIL	27	FAIL	39	FAIL
4	PASS	16	PASS	28	FAIL	40	PASS
5	FAIL	17	PASS	29	FAIL	41	FAIL
6	PASS	18	PASS	30	FAIL	42	PASS
7	PASS	19	PASS	31	PASS	43	FAIL
8	PASS	20	PASS	32	PASS	44	FAIL
9	PASS	21	PASS	33	PASS	45	FAIL
10	PASS	22	PASS	34	PASS		
11	PASS	23	PASS	35	FAIL		
12	PASS	24	PASS	36	FAIL		

4.1 Test 3.1 - Address Autoconfiguration and Duplicate Address Detection

4.1.1 Propósito

Verificar que un nodo puede inicializar apropiadamente en una red usando address autoconfiguration y comunicarse con otros nodos conectados.

4.1.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Dar tiempo a que se ejecute Stateless Address Autoconfiguration y DAD.
3. ConfigurarTN1 para que transmita un DAD Neighbor Solicitation (DAD NS) tal que Source = Unspecified address (::0) y Target Address = Link-local address del NUT.

4.1.3 Resultados esperados

2. El NUT debe ejecutar DAD en su tentative address para su interfaz enviando DupAddrDetectTransmits Neighbor Solicitations, cada un segundo. El NUT debe asignar el tentative address a su interfaz.
4. El NUT debe enviar un DAD Neighbor Advertisement (DAD NA) para su autoconfigured Link-local address.

4.1.4 Resultado obtenido

Wait DAD NA from NUT: timeout
NUT's address is not configured.
NUT sent DAD NS but not responds to DAD NS to Same address.
NG
Se esperaba un DAD NA, pero este no fue enviado

4.1.5 Solución

Ver test 3.15.

4.2 Test 3.3 - Part B: NUT receives DAD NS (target == NUT)

4.2.1 Propósito

Verificar que un nodo puede procesar las NS y los NA ejecutando Duplicate Address Detection mientras que el nodo también está ejecutando DAD.

4.2.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Luego de que TN1 recibe un DAD NS del NUT. Configurar TN1 para enviar un DAD NS con Target Address = tentative link-local address NUT's.
3. Dar tiempo a todos los dispositivos para que ejecuten Stateless address autoconfiguration y Duplicate Address Detection.
4. Enviar un NS del TN1 al Solicited-node multicast address del Link-local address del NUT's con Target Address = Link-local address del NUT
6. Enviar un NS del TN1 al Link-local address del NUT con Target Address = Link-local address del NUT.
8. Si el NUT es un Host, TR1 envía un RA con Prefix option.
9. TN1 envía un DAD NS tal que Source address = Unspecified address y Target address = Global Address del NUT.

4.2.3 Resultados esperados

3. The NUT debe recibir más DAD NS de lo esperado con Target address = tentative Link-local address. El NUT debe determinar que su tentative address es un duplicate and y no asignar la dirección a la interfaz y debe deshabilitar IP. El NUT NO debe enviar ningún RS si el NUT es un Host.
5. El NUT NO debe enviar un NA a su Autoconfigured Link-local address.
7. El NUT NO debe enviar un NA a su Autoconfigured Link-local address.
10. El NUT NO debe enviar un DAD NA a su Global address.

4.2.4 Resultado obtenido

El NUT must not transmit any Router Solicitations if el NUT is a Host.

Falló el paso 3 ya que se envió un RS que no era esperado.

4.2.5 Solución

Ver test 3.5

4.3 Test 3.5 - Part D: NUT receives DAD NA (target == NUT)

4.3.1 Propósito

Verificar que un nodo puede procesar las NS y los NA ejecutando Duplicate Address Detection mientras que el nodo también está ejecutando DAD.

4.3.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Luego de que TN1 recibe un DAD NS del NUT. Configurar TN1 para enviar un DAD NA con Target Address = tentative Link-local address del NUT y sin opción TLL.
3. Dar tiempo a todos los dispositivos para que ejecuten Stateless address autoconfiguration y Duplicate Address Detection.
4. Enviar un NS del TN1 al Solicited-node multicast address del Link-local address del NUT's con Target Address = Link-local address del NUT.
6. Enviar un NS del TN1 al Link-local address del NUT con Target Address = Link-local address del NUT.
8. Si el NUT es un Host, TR1 envía un RA con Prefix option.
9. TN1 envía un DAD NS tal que Source address = Unspecified address y Target address = Global Address del NUT.

4.3.3 Resultados esperados

3. The NUT debe recibir más DAD NS de lo esperado con Target address = tentative Link-local address. El NUT debe determinar que su tentative address es un duplicate and y no asignar la dirección a la interfaz y debe deshabilitar IP. El NUT NO debe enviar ningún RS si el NUT es un Host.
5. El NUT NO debe enviar un NA a su Autoconfigured Link-local address.
7. El NUT NO debe enviar un NA a su Autoconfigured Link-local address.
10. El NUT NO debe enviar un DAD NA a su Global address.

4.3.4 Resultado obtenido

El NUT must not transmit any Router Solicitations if el NUT is a Host.

Falló el paso 3 ya que se envió un RS que no era esperado.

4.3.5 Solución

El error en este test, era que no se cancelaba el timer de DAD, lo que llevaba a que se enviara un RS. Para resolver esto, se tiene que generar un caso particular (que eventualmente se puede mejorar) en la clase IPv6Protocol dentro de la función isForMe para aceptar los NA que provengan de la dirección FF02::1

```
Inet6Address addr =
(Inet6Address)Inet6Address.getByName("FF02:0:0:0:0:0:0:1");

if (((Inet6Address)header.getDestinationAddress()).equals(addr)){
	byte [] b = ns.getMessage().getBytes();
	if (b[54] == (byte)ICMPv6Type.NEIGHBOR_ADVERTISEMENT){
		return true;
	}
}
```

De esta forma se aceptan los NA cuya source address sea `FF02:0:0:0:0:0:0:1`
Este tipo de NA son los que se utilizan en el proceso de DAD, llamados *DAD NA*.

4.4 Test 3.14 - Part I: NUT receives valid DAD NS (Reserved Field)

4.4.1 Propósito

Verificar que un nodo puede ignorar las NS inválidas mientras ejecuta Duplicate Address Detection.

4.4.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Luego de que TN1 recibe un DAD NS del NUT. Configurar TN1 para enviar un NA con el campo Reserved = 0xFFFFFFFF.
3. Dar tiempo a todos los dispositivos para que ejecuten Stateless address autoconfiguration y Duplicate Address Detection.
4. Enviar un NS del TN1 al Solicited-node multicast address del Link-local address del NUT's con Target Address = Link-local address del NUT.
6. Enviar un NS del TN1 al Link-local address del NUT con Target Address = Link-local address del NUT.
8. Si el NUT es un Host, TR1 envía un RA con Prefix option.
9. TN1 envía un DAD NS tal que Source address = Unspecified address y Target address = Global Address del NUT.

4.4.3 Resultados esperados

3. El NUT debe ignorar los contenidos del campo Reserved. El NUT NO debe asignar la dirección a la interfaz y debe deshabilitar IP. El NUT NO debe enviar ningún RS si el NUT es un Host.
5. El NUT NO debe enviar un NA a su Autoconfigured Link-local address.
7. El NUT NO debe enviar un NA a su Autoconfigured Link-local address.
10. El NUT NO debe enviar un DAD NA a su Global address.

4.4.4 Resultado obtenido

El NUT must not transmit any Router Solicitations if el NUT is a Host.

Falló el paso 3 ya que se envió un RS que no era esperado.

4.4.5 Solución

Ver test 3.15

4.5 Test 3.15 - Part J: NUT receives valid DAD NS (contains TLL)

4.5.1 Propósito

Verificar que un nodo puede ignorar las NS inválidas mientras ejecuta Duplicate Address Detection.

4.5.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Luego de que TN1 recibe un DAD NS del NUT. Configurar TN1 para enviar un NA con opción TLL seteada al MAC address de TN1.
3. Dar tiempo a todos los dispositivos para que ejecuten Stateless address autoconfiguration y Duplicate Address Detection.
4. Enviar un NS del TN1 al Solicited-node multicast address del Link-local address del NUT's con Target Address = Link-local address del NUT.
6. Enviar un NS del TN1 al Link-local address del NUT con Target Address = Link-local address del NUT.
8. Si el NUT es un Host, TR1 envía un RA con Prefix option.
9. TN1 envía un DAD NS tal que Source address = Unspecified address y Target address = Global Address del NUT.

4.5.3 Resultados esperados

3. The NUT debe ignorar las opciones que no reconoce y continuar procesando el mensaje. El NUT NO debe asignar la dirección a la interfaz y debe deshabilitar IP. El NUT NO debe enviar ningún RS si el NUT es un Host.
5. El NUT NO debe enviar un NA a su Autoconfigured Link-local address.
7. El NUT NO debe enviar un NA a su Autoconfigured Link-local address.
10. El NUT NO debe enviar un DAD NA a su Global address.

4.5.4 Resultado obtenido

El NUT must not transmit any Router Solicitations if el NUT is a Host.

Falló el paso 3 ya que se envió un RS que no era esperado.

4.5.5 Solución

Estos tests tuvieron la misma resolución que fue cambiar el lugar de la cancelación del timer para DAD en el método:

```
rcvICMPv6NeighborSolicitation(ICMPv6Header, IPv6ProtocolHeader,
IPv6NetParameters, String, NetConfigs);
```

La llamada a la cancelación del timer es la siguiente:

```
netParams.cancelDupAddressTimer(jth,icmpNS.targetAddress,false);
```

Se cambio el lugar dentro del código hacia el final del método

4.6 Test 3.27 - Part A: Unicast Autoconfigured Address – Global

4.6.1 Propósito

Verificar que un nodo ejecuta DAD en su Autoconfigured Unicast Address.

4.6.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Configurar TR1 para enviar un RA con prefijo “X” y con el campo Valid Lifetime = 40 (seg.). Si NUT es un Router, configurar un Global address con prefijo “X”.
3. Dar tiempo al NUT para que ejecute Stateless address autoconfiguration y Duplicate Address Detection.
4. Enviar un DAD NS desde TN1 con Target Address = Global address del NUT.

4.6.3 Resultados esperados

3. El NUT debe ejecutar DAD en su Global address tentativo para su interfaz, enviando DupAddrDetectTransmits NS´s, cada RetransTimer segundos. El NUT debe asignar el Global address tentativo a la interfaz.
5. El NUT debe enviar un DAD NA a su Autoconfigured Global address.

4.6.4 Resultado obtenido

NUT had not transmitted DAD NS for Global address.

4.6.5 Solución

Ver test 3.45

4.7 Test 3.28 - Part B: Unicast Autoconfigured Address Prefix ending in zero valued fields

4.7.1 Propósito

Verificar que un nodo ejecuta DAD en su Autoconfigured Unicast Address.

4.7.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Configurar TR1 para enviar un RA con prefijo “8000:0000::/64” y con el campo Valid Lifetime = 40 (seg.). Si NUT es un Router, configurar un Global address con prefijo “8000:0000::/64”.
3. Dar tiempo al NUT para que ejecute Stateless address autoconfiguration y Duplicate Address Detection.
4. Enviar un DAD NS desde TN1 con Target Address = Global address del NUT.

4.7.3 Resultados esperados

3. El NUT debe ejecutar DAD en su Global address tentativo para su interfaz, enviando DupAddrDetectTransmits NS’s, cada RetransTimer segundos. El NUT debe asignar el Global address tentativo a la interfaz.
5. El NUT debe enviar un DAD NA a su Autoconfigured Global address.

4.7.4 Resultado obtenido

NUT had not transmitted DAD NS for Global address.

4.7.5 Solución

Ver test 3.45

4.8 Test 3.29 - Part C: Unicast Autoconfigured Address Site-Local

4.8.1 Propósito

Verificar que un nodo ejecuta DAD en su Autoconfigured Unicast Address.

4.8.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Configurar TR1 para enviar un RA con prefijo “FEC0::/64” y con el campo Valid Lifetime = 40 (seg.). Si NUT es un Router, configurar un Global address con prefijo “FEC0::/64”.
3. Dar tiempo al NUT para que ejecute Stateless address autoconfiguration y Duplicate Address Detection.
4. Enviar un DAD NS desde TN1 con Target Address = Global address del NUT.

4.8.3 Resultados esperados

3. El NUT debe ejecutar DAD en su Global address tentativo para su interfaz, enviando DupAddrDetectTransmits NS’s, cada RetransTimer segundos. El NUT debe asignar el Global address tentativo a la interfaz.
5. El NUT debe enviar un DAD NA a su Autoconfigured Global address.

4.8.4 Resultado obtenido

NUT had not transmitted DAD NS for Global address.

4.8.5 Solución

Ver test 3.45

4.9 Test 3.30 - Address Lifetime Expiry (Hosts Only)

4.9.1 Propósito

Verificar que un host puede manejar correctamente direcciones inválidas y expiradas.

4.9.2 Procedimiento

1. Inicializar todos los dispositivos.
2. Configurar TR1 para enviar un RA con prefijo “X” y con el campo Valid Lifetime = 40 (seg).
3. Dar tiempo al NUT para que ejecute Stateless address autoconfiguration y Duplicate Address Detection.
4. Configurar TR1 para enviar un NS para Address resolution con Target address = Global address del HUT para el prefijo “X”.
5. Esperar 35 segundos.
6. Repetir el paso 4.
7. Esperar 10 segundos.
8. Configurar TR1 para enviar un NS para Address resolution con Target address = Global address del HUT para el prefijo “X”.

4.9.3 Resultados esperados

Step 3: El HUT debe enviar un DAD NS a su Autoconfigured global address.
Step 4: El HUT debe enviar un NA a su Autoconfigured global address.
Step 6: El HUT debe enviar un NA a su Autoconfigured global address.
Step 8: El HUT NO debe enviar un NA a su Autoconfigured global address usando el prefijo “X”.

4.9.4 Resultado obtenido

Received packet count=1
Wait DAD NA from NUT: timeout
NUT's address is not configured.
NUT did not assign Global address.

4.9.5 Solución

Ver test 3.45

4.10 Test 3.36 - Part E: prefix length > 128 bits

4.10.1 Propósito

Verificar que un host procesa la opción Prefix Information en un Router Advertisement.

4.10.2 Procedimiento

1. El HUT debe tener un identificador de interfaz de largo > 0. TR1 envía un Router Advertisement A con Prefix Length = 128.
2. Configurar TR1 para transmitir un NS para Address resolution con Target address = Global address del HUT para el prefijo “X”.

4.10.3 Resultados esperados

1. El HUT debe ignorar la opción de Prefix Information Option y NO formar una dirección usando el prefijo “X”.
3. El HUT NO debe enviar un NA para su Autoconfigured global address con prefijo “X”.

4.10.4 Resultado obtenido

```
Received packet count=1
Wait NA from NUT: timeout
NUT's address is not configured.
NUT had not assign Global address.
NG
```

Se recibió un paquete no esperado.

4.10.5 Solución

Modificar la clase:
`ip4jvm.net.applications.NeighborDiscovery`

Método

```
prefixCorrect(ICMPv6PrefixInfoOpt);
```

Agregar la siguiente línea de código al método:

```
if (prefixInfOpt.getPrefixLength()> 128) return false;
```

Esto valida que el largo del prefijo no sea mayor que 128, que en este caso se debe descartar el prefijo

4.11 Test 3.37 - Part F: prefix length < 64 bits

4.11.1 Propósito

Verificar que un host procesa la opción Prefix Information en un Router Advertisement.

4.11.2 Procedimiento

1. El HUT debe tener un identificador de interfaz de largo > 0. TR1 envía un Router Advertisement A con Prefix Length = 0.
2. Configurar TR1 para transmitir un NS para Address resolution con Target address = Global address del HUT para el prefijo “X”.

4.11.3 Resultados esperados

1. El HUT debe ignorar la opción de Prefix Information Option y NO formar una dirección usando el prefijo “X”.
3. El HUT NO debe enviar un NA para su Autoconfigured global address con prefijo “X”.

4.11.4 Resultado obtenido

```
Received packet count=1
Wait NA from NUT: timeout
NUT's address is not configured.
NUT had not assign Global address.
NG
```

Se recibió un paquete no esperado.

4.11.5 Solución

Modificar la clase:

```
ip4jvm.net.applications.NeighborDiscovery
```

Método
`prefixCorrect(ICMPv6PrefixInfoOpt);`

Agregar la siguiente línea de código al método:

```
if (prefixInfOpt.getPrefixLength()<64) return false;
```

Esto valida que el largo del prefijo no sea menor a 64, que en este caso se debe descartar el prefijo

4.12 Test 3.39 - Part H: Valid Lifetime is zero

4.12.1 Propósito

Verificar que un host procesa la opción Prefix Information en un Router Advertisement.

4.12.2 Procedimiento

1. TR1 envía un Router Advertisement A con el campo Valid Lifetime = 0.
2. Configurar TR1 para transmitir un NS para Address resolution con Target address = Global address del HUT para el prefijo “X”.

4.12.3 Resultados esperados

1. El HUT debe ignorar la opción de Prefix Information Option y NO formar una dirección usando el prefijo “X”.
3. El HUT NO debe enviar un NA para su Autoconfigured global address con prefijo “X”.

4.12.4 Resultado obtenido

```
Received packet count=1
Received NA from NUT: NA_from_NUT_GA0Tgt
NUT assigned the address to the interface.
NUT assigned Global address to the interface.
NG
```

Falla el paso 3, se envía el NA a pesar de que no debería ser enviado.

4.12.5 Solución

Modificar la clase:

```
ip4jvm.net.applications.NeighborDiscovery
```

Método

```
prefixCorrect(ICMPv6PrefixInfoOpt);
```

Agregar la siguiente línea de código al método:

```
if (prefixInfOpt.getValidLifeTime() == 0) return false;
```

Esto valida que el tiempo válido del prefijo no sea 0, ya que en este caso se debe descartar el prefijo.

4.13 Test 3.41 - Part J: Valid Lifetime is 0xffffffff

4.13.1 Propósito

Verificar que un host procesa apropiadamente la opción Prefix Information de un RA.

4.13.2 Procedimiento

1. TR1 envía un Router Advertisement A con el campo Valid Lifetime = 0xffffffff.
2. Configurar TR1 para que envié un NS para Address resolution con Target address = Global address para el prefijo “X” del HUT.

4.13.3 Resultados esperados

3. El HUT debe procesar la opción Prefix Information y formar una dirección para el prefijo “X”. El HUT debe enviar un NA para su Autoconfigured global address con prefijo “X”.

4.13.4 Resultado obtenido

NUT had not transmitted DAD NS for Global address.
NG

No se transmitió el DAD NS para la dirección global.

4.13.5 Solución

Ver test 3.45

4.14 Test 3.43 - Part B: Prefix Lifetime greater than 2 hours

4.14.1 Propósito

Verificar que un host actualiza apropiadamente su Address List al recibir una opción Prefix Information.

4.14.2 Procedimiento

1. TR1 envía un Router Advertisement A con el campo Valid Lifetime = 3hrs.
2. TR1 envía un Router Advertisement con un prefijo del prefijo Global de TR1 y el campo Valid Lifetime = 2hrs 30s.
3. Esperar 2hrs 45 segundos.
4. Configurar TR1 para que envié un NS para Address resolution con Target address = Global address para el prefijo “X” del HUT.

4.14.3 Resultados esperados

5. El HUT NO debe enviar un NA para su Autoconfigured global address con prefijo “X”.

4.14.4 Resultado obtenido

Received packet count=0
Wait NA from NUT: timeout
NUT's address is not configured.
The address is not available.
NG

4.14.5 Solución

En lugar de detenerse a los 7205 segundos, el timer continúa durante 7514 segundos, lo que provoca que la dirección ya no esté disponible tras este tiempo. Esto se debe a un tema del hardware virtual de la VM, se debe dar más memoria al TN de manera que el timer funcione de manera correcta.
Se elevó la memoria del TN a 768 Mb, y tras correr el test nuevamente, pasa.

4.15 Test 3.44 - Part C: Prefix Lifetime less than the Stored Lifetime and the Stored Lifetime is less than 2 hours

4.15.1 Propósito

Verificar que un host actualiza apropiadamente su Address List al recibir una opción Prefix Information.

4.15.2 Procedimiento

1. TR1 envía un Router Advertisement A con el campo Valid Lifetime = 60 (seg.).
2. TR1 envía un Router Advertisement con un prefijo del prefijo Global de TR1 y Valid Lifetime = 30 (seg.).
3. Esperar 35 segundos.
4. Configurar TR1 para que transmita un NS para address resolution con Target address = Global address del HUT para el prefijo “X”.

4.15.3 Resultados esperados

5. El HUT debe enviar un NA al autoconfigured global address con prefijo “X”.

4.15.4 Resultado obtenido

Received packet count=0
Wait NA from NUT: timeout
NUT's address is not configured.
The address is not available.
NG

Falló el paso 5: se debería recibir un NA, pero no se recibe ningún paquete.

4.15.5 Solución

Ver test 3.45

4.16 Test 3.45 - Part D: Prefix Lifetime less than 2 hours and the Stored Lifetime is greater than 2 hours

4.16.1 Propósito

Verificar que un host actualiza apropiadamente su Address List al recibir una opción Prefix Information.

4.16.2 Procedimiento

1. TR1 envía un Router Advertisement A con el campo Valid Lifetime = 2hrs 30s.
2. TR1 envía un Router Advertisement con un prefijo del prefijo Global de TR1 y Valid Lifetime = 10 (seg.).
3. Esperar 11 segundos.
4. Configurar TR1 para que transmita un NS para address resolution con Target address = Global address del HUT para el prefijo “X”.
5. Esperar 2hrs 15 segundos.
6. Configurar TR1 para que transmita un NS para address resolution con Target address = Global address del HUT para el prefijo “X”.

4.16.3 Resultados esperados

7. El prefijo “X” debe vencer su Valid Lifetime. El HUT NO debe enviar un NA para su Autoconfigured global address con prefijo “X”.

4.16.4 Resultado obtenido

Received packet count=0
Wait NA from NUT: timeout
NUT's address is not configured.
The address is not available. NG

4.16.5 Solución

Estos tests fallaban, ya que no se enviaba el DAD NS con el Global address cuando se recibía el DAD NS (ver tests). Se resolvió modificando la clase:

`ip4jvm.net.applications.NeighborDiscovery`

Método:

```
rcvICMPv6RouterAdvertisement(ICMPv6Header, IPv6ProtocolHeader,
IPv6NetParameters, String, NetConfigs);
```

Agregando el siguiente código:

```
Inet6Address dstAddress;
try {
       dstAddress= IPv6Utils.solicitedNodeMulticastAddress(autoIP);
}catch(Exception e){
       System.err.println(e);
       dstAddress = null;
}
MACAddress targetLL= (MACAddress)
   ((IPv6NetParameters)stack.getNetParameters()).getLinkLayerAddress(jth);
snd(ICMPv6Type.NEIGHBOR_SOLICITATION,
    autoIP, null, dstAddress, IPv6Utils.unspecifiedAddress(), jth, null);
```

Esto envía el DAD NS con el Global address.

5 Section IV - RFC 1981 - Path MTU Discovery for IPv6

Los siguientes tests cubren la especificación de Path MTU Discovery para IPv6, como se describe en el RFC 1981.

El protocolo Path MTU Discovery es una técnica para descubrir dinámicamente el PMTU de un camino. La idea básica es que el nodo fuente inicialmente asume que el PMTU de un path es el MTU (conocido). Si uno de los paquetes enviados por el camino es demasiado grande, se descarta el paquete y se devuelve un mensaje ICMPv6 Packet Too Big.

Al recibir dicho mensaje, el nodo reduce el PMTU para el camino; utilizando el que viene en el mensaje Packet Too Big. El proceso termina cuando la estimación es menor o igual que el PMTU actual.

Estado inicial:

1	-	5	FAIL	9	FAIL	13	FAIL
2	PASS	6	FAIL	10	FAIL	14	FAIL
3	PASS	7	FAIL	11	FAIL	15	FAIL
4	PASS	8	FAIL	12	FAIL	16	FAIL

12 tests fallidos de 16.

5.1 Test 4.5 - Stored PMTU

5.1.1 Propósito

Verificar que un nodo puede almacenar información para el proceso de PMTU para múltiples destinos.

5.1.2 Procedimiento

1. TN1 envía un Echo Request al NUT de tamaño 1500 bytes.
2. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.
3. TR1 reenvía un Echo Request de TN3 al NUT de tamaño 1500 bytes.
5. TR1 envía un mensaje Packet Too Big al NUT a TN2, con el campo MTU = 1400.
6. TN1 envía un Echo Request al NUT de tamaño 1500 bytes.
7. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.
8. TR1 reenvía un Echo Request de TN3 al NUT de tamaño 1500 bytes.
10. TR1 envía un mensaje Packet Too Big al NUT a TN2, con el campo MTU = 1280.
11. TN1 envía un Echo Request al NUT de tamaño 1500 bytes.
12. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.
13. TR1 reenvía un Echo Request de TN3 al NUT de tamaño 1500 bytes.

5.1.3 Resultados esperados

4. El NUT debe enviar 3 Echo Reply, uno a TN1, uno a TN2, y uno a TN3.
9. El NUT debe responder a los 3 Echo Request. Los Echo Reply a TN1 y TN3 no debe ser más grande que 1500 bytes. No se debe fragmentar estos paquetes. El NUT debe fragmentar el Echo Reply a TN2 y cada fragmento no debe ser más grande que 1400 bytes.
14. El NUT debe responder a los 3 Echo Request. El Echo Reply a TN1 no debe ser más grande que 1500 bytes. No se debe fragmentar este paquete. El NUT debe fragmentar el Echo Reply a TN2 y cada fragmento no debe ser más grande que 1400 bytes. El NUT debe fragmentar el Echo Reply a TN3 y cada fragmento no debe ser más grande que 1280 bytes.

5.1.4 Resultado obtenido

Fallan los pasos 9 y 14.

5.1.5 Solución

Ver test 4.14

5.2 Test 4.6 - Non-zero ICMPv6 Code

5.2.1 Propósito

Verificar que un nodo procesa un mensaje Packet Too Big con el campo Code != 0.

5.2.2 Procedimiento

1. TR1 reenvía un Echo Request de TN2 al NUT.
3. TR1 envía un mensaje Packet Too Big al NUT, que contiene el campo Code = 0xFF (inválido) y el campo MTU=1280.
4. TR1 reenvía un Echo Request de TN2 al NUT.

5.2.3 Resultados esperados

2. El NUT debe responder al Echo Request usando a TR1 como “first hop”.
5. El NUT debe fragmentar la respuesta al Echo Request usando TR1 como “first hop”, lo que indica que el NUT ignoro el campo Code inválido y proceso el mensaje Packet Too Big. Los fragmentos no deben ser más grandes que 1280 bytes.

5.2.4 Resultado obtenido

Falla el paso 5.

5.2.5 Solución

Ver test 4.14

5.3 Test 4.7 - Reduce PMTU On-link

5.3.1 Propósito

Verificar que un nodo procesa un mensaje Packet Too Big indicando una reducción en el PMTU para un destino On-link (conocido).

5.3.2 Procedimiento

1. TR1 envía un Echo Request de 1500 bytes al NUT.
3. TR1 envía un mensaje Packet Too al NUT con el campo MTU = 1280.
4. TR1 envía un Echo Request fragmentado de 1500 bytes al NUT. Los paquetes fragmentados no son más grandes que 1280 bytes.
6. Re-bootear el NUT.
7. Repetir los pasos 1 al 5, enviando un Echo Request al NUT en los pasos 1 y 4.

5.3.3 Resultados esperados

2. El NUT debe responder al Echo Request.
5. El NUT debe fragmentar correctamente la respuesta al Echo Request, indicando que el NUT procesó el mensaje Packet Too Big. Los paquetes fragmentados no son más grandes que 1280 bytes.

5.3.4 Resultado obtenido

Falla el paso 5.

5.3.5 Solución

Ver test 4.14

5.4 Test 4.8 - Reduce PMTU Off-link

5.4.1 Propósito

Verificar que un nodo procesa un mensaje Packet Too Big indicando una reducción en el PMTU para un destino Off-link (desconocido).

5.4.2 Procedimiento

1. TR1 envía un Echo Request de TN2 al NUT.
3. TR1 envía un mensaje Packet Too Big al NUT con el campo MTU = 1400.
4. TR1 envía un Echo Request de TN2 al NUT de tamaño 1500 bytes.
6. TR1 envía otro mensaje Packet Too Big con el campo MTU = 1280.
7. TR1 envía un Echo Request de TN2 al NUT de tamaño 1500 bytes.

5.4.3 Resultados esperados

2, El NUT debe responder al Echo Request usando TR1 como “first hop”.
5. El NUT debe fragmentar la respuesta. Los fragmentos no pesan más de 1400 bytes.
8. El NUT debe fragmentar la respuesta. Los fragmentos no pesan más de 1280 bytes.

5.4.4 Resultado obtenido

Fallaron los pasos 5 y 8.

5.4.5 Solución

Aparte de la solución propuesta por el test 4.14, se debe agregar lo siguiente.
No se actualiza el valor del MTU del link (cuando no existe entrada en el Destination Cache)
Se debe reducir el MTU del link ya que no existe la entrada en el cache.

```
netParams.setLinkMtu(jth, icmpPTB.mtu);
```

Al no existir la entrada en el cache, el MTU que se manejaba era el ETH_MTU=1500.

5.5 Test 4.9 - Part A: MTU equal to 56

5.5.1 Propósito

Verificar que un nodo no reduce el PMTU por debajo del IPv6 minimum MTU (1280).

5.5.2 Procedimiento

1. TR1 envía un Echo Request desde TN2 hacia el NUT. El tamaño es de 1280 bytes.
3. TR1 envía un mensaje Packet Too Big al NUT, que contiene un campo MTU=56.
4. TR1 envía un Echo Request desde TN2 hacia el NUT. El tamaño es de 1280 bytes.

5.5.3 Resultados esperados

2. El NUT debe responder al Echo Request.
5. El NUT debe responder al Echo Request, y no reducir el PMTU debajo de IPv6 minimum link MTU (1280). En lugar de eso, debe incluir un Fragment Header en el Echo Reply. El Echo Reply debe tener un tamaño de 1280 bytes.

5.5.4 Resultado obtenido

Cannot receive Echo Reply. NG

Falla el paso 5, ya que no se incluye el Fragment Header dentro de la respuesta.

5.5.5 Solución

Ver test 4.10

5.6 Test 4.10 - Part B: MTU equal to 1279

5.6.1 Propósito

Verificar que un nodo no reduce el PMTU por debajo del IPv6 minimum MTU (1280).

5.6.2 Procedimiento

1. TR1 envía un Echo Request desde TN2 hacia el NUT. El tamaño es de 1280 bytes.
3. TR1 envía un mensaje Packet Too Big al NUT, que contiene un campo MTU=1279.
4. TR1 envía un Echo Request desde TN2 hacia el NUT. El tamaño es de 1280 bytes.

5.6.3 Resultados esperados

2. El NUT debe responder al Echo Request.
5. El NUT debe responder al Echo Request, y no reducir el PMTU debajo de IPv6 minimum link MTU (1280). En lugar de eso, debe incluir un Fragment Header en el Echo Reply. El Echo Reply debe tener un tamaño de 1280 bytes.

5.6.4 Resultado obtenido

Cannot receive Echo Reply. NG

Falla paso 5. Ídem al test 4.9.

5.6.5 Solución

Para estos tests hubo que modificar el manejo de MTU ya que según la regla
"Si el MTU recibido es menor a MIN_MTU=1280, se debe setear el MTU a 1280."

Para esto se agrego un booleano que checkea el mtu < MIN_MTU

```
if (icmpPTB.mtu<IPv6Protocol.MIN_MTU)
      mtulessthanMINMTU = true;
```

Así luego al obtener el valor del MTU, si la variable esta en trae, se maneja el valor de mtu = MIN_MTU y se agrega el Fragment Header que se pide en el paso 5.

5.7 Test 4.11 - Part A: MTU increase

5.7.1 Propósito

Verificar que un nodo no incrementa su estimación de PMTU al recibir un mensaje Packet Too Big.

5.7.2 Procedimiento

1. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.
3. TR1 envía un mensaje Packet Too Big al NUT. El campo MTU = 1304.
4. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.
6. TR1 envía un mensaje Packet Too Big al NUT. El campo MTU = 1500.
7. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.

5.7.3 Resultados esperados

2. El NUT debe responder al Echo Request usando TR1 como “first hop”.
5. El NUT debe fragmentar la respuesta al Echo Request indicando que el NUT proceso el mensaje Packet Too Big.
8. El NUT debe fragmentar la respuesta al Echo Request tal que el tamaño de los fragmentos no es mayor a 1304 bytes. El NUT NO debe procesar el segundo mensaje Packet Too Big que indica un incremento en el PMTU.

5.7.4 Resultado obtenido

Fallan los pasos 5 y 8.

5.7.5 Solución

Ver test 4.14

5.8 Test 4.12 - Part B: MTU equal to 0x1FFFFFFF

5.8.1 Propósito

Verificar que un nodo no incrementa su estimación de PMTU al recibir un mensaje Packet Too Big.

5.8.2 Procedimiento

1. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.
3. TR1 envía un mensaje Packet Too Big al NUT. El campo MTU = 1304.
4. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.
6. TR1 envía un mensaje Packet Too Big al NUT. El campo MTU = 0x1FFFFFFF.
7. TR1 reenvía un Echo Request de TN2 al NUT de tamaño 1500 bytes.

5.8.3 Resultados esperados

2. El NUT debe responder al Echo Request usando TR1 como “first hop”.
5. El NUT debe fragmentar la respuesta al Echo Request indicando que el NUT proceso el mensaje Packet Too Big.
8. El NUT debe fragmentar la respuesta al Echo Request tal que el tamaño de los fragmentos no es mayor a 1304 bytes. El NUT NO debe procesar el segundo mensaje Packet Too Big que indica un incremento en el PMTU.

5.8.4 Resultado obtenido

Fallan los pasos 5 y 8.

5.8.5 Solución

Ver test 4.14

5.9 Test 4.13 - Router Advertisement with MTU Option (Hosts Only)

5.9.1 Propósito

Verificar que un host procesa correctamente un RA con opción MTU.

5.9.2 Procedimiento

1. TR1 envía un Echo Request desde TN2 al HUT con tamaño 1500 btyes.
3. TR1 envía un Router Advertisement con opción MTU=1280.
4. TR1 envía un Echo Request fragmentado desde TN2 al HUT con tamaño total (re ensamblado) de 1500 bytes.

5.9.3 Resultados esperados

2. El HUT debe responder al Echo Request. El HUT NO debe fragmentar el Echo Reply.
5. El HUT debe actualizar su Link MTU para TR1 a 1280. El HUT debe fragmentar la respuesta al Echo Request, indicando que el HUT cambio el PMTU. Los paquetes fragmentados no deben medir más de 1280 bytes.

5.9.4 Resultado obtenido

Cannot receive Echo Reply
NG

Falla paso 5.

5.9.5 Solución

Además de la solución 2.14

Dentro de la recepción de un RA, método: `rcvICMPv6RouterAdvertisement`
Se agregó:

```
netParams.setLinkMtu(jth,linkMTU);
```

Con esto se setea el MTU del link para que el test sea exitoso

5.10 Test 4.14 - Checking For Increase in PMTU

5.10.1 Propósito

Verificar que un nodo espera la cantidad de tiempo apropiada para chequear un incremento en el PMTU.

5.10.2 Procedimiento

1. TR1 envía un Echo Request de TN2 al NUT.
3. TR1 envía un mensaje Packet Too Big al NUT. El campo MTU = 1304.
4. TR1 envía un Echo Request de TN2 al NUT.
6. TR1 envía un Echo Request de TN2 cada 30 seg durante 5 minutos.

5.10.3 Resultados esperados

2. El NUT debe responder al Echo Request.
5. El NUT debe fragmentar correctamente la respuesta al Echo Request, indicando que proceso el mensaje Packet Too Big de TR1. Los paquetes fragmentados NO deben ser más grandes que 1304 bytes.
7. El NUT NO debe transmitir paquetes de más de 1304 bytes durante 5 minutos.

5.10.4 Resultado obtenido

Falla paso 5.

5.10.5 Solución

Para estos tests, se modificó la forma de salvar el MTU, seteando todos los valores del Destination Cache, de los vecinos asociados al router que envío el mensaje PACKET_TOO_BIG, ya que no se podía obtener el MTU de cada link.

5.11 Test 4.15 - Multicast Destination - One Router

5.11.1 Propósito

Verificar que un nodo elige el PMTU para destinos multicast.

5.11.2 Procedimiento

1. El NUT envía un Echo Request de 1500 bytes y Destination = FF1E::1:2.
2. TR1 envía un mensaje Packet Too Big al NUT con MTU = 1450.
3. Se envía el mismo paquete del paso 1.
5. TR1 envía un mensaje Packet Too Big al NUT con MTU = 1400.
6. Se envía el mismo paquete del paso 1.
8. El NUT envía un Echo Request de 1400 bytes y Destination = FF1E::1:2.
9. TR1 envía un mensaje Packet Too Big al NUT con MTU = 1300.
10. Se envía el mismo paquete del paso 1.
12. TR1 envía un mensaje Packet Too Big al NUT con MTU = 1350.
13. Se envía el mismo paquete del paso 1.

5.11.3 Resultados esperados

4. El NUT debe fragmentar el Echo Request a FF1E::1:2, indicando que proceso los mensajes Packet Too Big de TR1. Los paquetes fragmentados no deben ser más grandes que 1450 bytes.
7. El NUT debe fragmentar el Echo Request a FF1E::1:2, indicando que proceso los mensajes Packet Too Big de TR1. Los paquetes fragmentados no deben ser más grandes que 1400 bytes.
11. El NUT debe fragmentar el Echo Request a FF1E::1:2, indicando que proceso los mensajes Packet Too Big de TR1. Los paquetes fragmentados no deben ser más grandes que 1300 bytes.
14. El NUT debe fragmentar el Echo Request a FF1E::1:2, indicando que proceso los mensajes Packet Too Big de TR1. Los paquetes fragmentados no deben ser más grandes que 1300 bytes.

5.11.4 Resultado obtenido

No se pueden enviar pings de un cierto tamaño.

5.11.5 Solución

Ver test 4.16

5.12 Test 4.16 - Multicast Destination - Two Routers

5.12.1 Propósito

Verificar que un nodo elige el PMTU para destinos multicast cuando recibe mensajes PTB para más de un router.

5.12.2 Procedimiento

1. El NUT envía un Echo Request de 1500 bytes y Destination = FF1E::1:2.
2. TR1 envía un mensaje Packet Too Big al NUT con MTU = 1480.
3. Se envía el mismo paquete del paso 1.
5. TR1 envía un mensaje Packet Too Big al NUT con MTU = 1440.
6. Se envía el mismo paquete del paso 1.
8. TR1 envía un mensaje Packet Too Big al NUT con MTU = 1400.
9. TR2 envía un mensaje Packet Too Big al NUT con MTU = 1360.
10. Se envía el mismo paquete del paso 1.
12. TR1 envía un mensaje Packet Too Big al NUT con MTU = 1280.
13. TR2 envía un mensaje Packet Too Big al NUT con MTU = 1320.
14. Se envía el mismo paquete del paso 1.

5.12.3 Resultados esperados

4. El NUT debe fragmentar el Echo Request a FF1E::1:2, indicando que proceso los mensajes Packet Too Big de TR1. Los paquetes fragmentados no deben ser más grandes que 1480 bytes.
7. El NUT debe fragmentar el Echo Request a FF1E::1:2, indicando que proceso los mensajes Packet Too Big de TR1. Los paquetes fragmentados no deben ser más grandes que 1440 bytes.
11. El NUT debe fragmentar el Echo Request a FF1E::1:2, indicando que proceso los mensajes Packet Too Big de TR1 y TR2. Los paquetes fragmentados no deben ser más grandes que 1360 bytes.
15. El NUT debe fragmentar el Echo Request a FF1E::1:2, indicando que proceso los mensajes Packet Too Big de TR1 y TR2. Los paquetes fragmentados no deben ser más grandes que 1280 bytes.

5.12.4 Resultado obtenido

No se pueden enviar pings de un cierto tamaño.

5.12.5 Solución

Estos dos tests requerían que los ping enviados desde el NUT tuvieran un cierto tamaño, cosa que no estaba implementada en Applications.Ping6.

Para que los tests funcionaran se agrego el método:

```
snd(int seq, int id, Inet6Address dst, Inet6Address src, String jth, int size)
```

Este método se comporta de manera similar al método snd ya implementado solo que agregando (y haciendo uso de) la variable size que nos permite especificar el tamaño del paquete.

Cuando se prueban los tests se debe correr la clase de test modificado (testPing6v3) que permite especificar aparte del destination address y de la cantidad de pings que se hace; especificar el tamaño del paquete.

Ejemplo:

El test nos pide:

lio_start
ping6> Press enter key, and excute the following commands within 5 seconds.
ping6> Do ``ping6 DataSize= 1452, Send Count= 1, Interface= eth0, Destination= FF1E::1:2"
lio_stop

Desde el lado del NUT debemos ingresar en la consola (corriendo testPingv3)

```
> ping6 eth0 1 FF1E::1:2 1452
```

Esto específica que el ping se hace por la interfaz eth0, 1 vez, a la dirección FF1E::1:2, y su tamaño es de 1452 bytes.

6 Section V - RFC 4443 - ICMPv6

Los siguientes tests cubren la especificación del Internet Control Message Protocol para IP version 6, como se describe en el Request For Comments 4443.

El Protocolo de Mensajes de Control de Internet o ICMP (por sus siglas de Internet Control Message Protocol) es el sub protocolo de control y notificación de errores del Protocolo de Internet (IP). Como tal, se usa para enviar mensajes de error, indicando por ejemplo que un cierto campo es incorrecto (Parameter Problem) o que un router o host no puede ser localizado (Destination Unreachable).

Los tres mensajes de error que cubren estos tests son:

- Destination Unreachable
- Time Exceeded
- Parameter Problem

Estado inicial:

1	-	6	FAIL	11	PASS	16	PASS	21	FAIL
2	FAIL	7	FAIL	12	FAIL	17	PASS	22	FAIL
3	FAIL	8	FAIL	13	FAIL	18	FAIL	23	PASS
4	PASS	9	PASS	14	FAIL	19	PASS	24	PASS
5	FAIL	10	FAIL	15	FAIL	20	FAIL	25	FAIL

16 tests fallidos de 25.

6.1 Tests que no fallaron por problemas en el código

Test 5.2

Test correcto que inicialmente falló por problemas de operabilidad

Para que este test funcione desde el NUT corriendo el stack, se debe enviar un echo Request.

Tests 5.3, 5.5, 5.6, 5.7, 5.8, 5.10, 5.14. 5.22

Tests correctos que fallaron por problemas de configuración en el lado del TN.

6.2 Test 5.12 - Erroneous Header Field (Parameter Problem Generation)

6.2.1 Propósito

Verificar que un nodo genera apropiadamente un Parameter Problem Message cuando se encuentra un Erroneous Header Field.

6.2.2 Procedimiento

1. TN1 envía un Echo Request al NUT tal que:
   - Source Address = Global address del TN.
   - Destination Address = Global address del NUT.

6.2.3 Resultados esperados

2. El NUT debe descartar el Echo Request (y no enviar un Echo Reply).
El NUT debe enviar un Parameter Problem Meesage al TN tal que:
   - Code = 0 (Erroneous Header Field)
   - Pointer = 0x04 (offset del campo Payload Length)
   - Source Address = Destination Address del Echo Request del paso 1.

6.2.4 Resultado obtenido

TN can not receive ICMPv6 error message from NUT

Se debería enviar un ICMP error message código 0 (Erroneous Header Field).

6.2.5 Solución

Ver test 1.53.

6.3 Test 5.13 - Erroneous Header Field (Parameter Problem Generation)

6.3.1 Propósito

Verificar que un nodo genera apropiadamente un Parameter Problem Message cuando se encuentra un Unrecognized Next Header type.

6.3.2 Procedimiento

1. Se envía un Echo Request que tiene un Destination Option Header con un valor de Next Header = 128.

6.3.3 Resultados esperados

2. El NUT debe descartar el Echo Request (y no enviar un Echo Reply).
El NUT debe enviar un Parameter Problem Meesage al TN tal que:
Code = 1 (Unrecognized Next Header type)
Pointer = 0x28 (offset del campo Next Header)
Source Address = Destination Address del Echo Request del paso 1.

6.3.4 Resultado obtenido

Cannot receive ICMPv6 Parameter Problem Message

6.3.5 Solución

Ver test 1.8

6.4 Test 5.18 - Part A: UDP Port Unreachable (Routers and Hosts)

6.4.1 Propósito

Verificar que un nodo maneja apropiadamente la recepción de una condición de error causada por un paquete con un Source address que no identifica a un nodo único.

6.4.2 Procedimiento

1. Se envía un Echo Request tal que:
   - Source address = Unespecified address
   - Destination address = Global address del NUT
   - Port number = 9000

6.4.3 Resultados esperados

2. El NUT no debe enviar un Destination Unreachable Error Message al TN cuando recibe un paquete UDP para un puerto no alcanzable (unreachable).

6.4.4 Resultado obtenido

TN receive ICMPv6 error message from NUT

El NUT no debería mandar un mensaje Destination Unreachable si el source address es unespecified (::), sin embargo este mensaje es enviado

6.4.5 Solución

Ver test 5.21.

6.5 Test 5.20 - Part C: Echo Request Reassembly Timeout (Routers and Hosts)

6.5.1 Propósito

Verificar que un nodo maneja apropiadamente la recepción de una condición de error causada por un paquete con un Source address que no identifica a un nodo único.

6.5.2 Procedimiento

1. Se envía el primer fragmento de un paquete tal que:
   - Source address = Unespecified address
   - Destination address = Link-local address del NUT
   - M flag = 1, Offset = 0

6.5.3 Resultados esperados

2. El NUT no debe enviar un Time Exceeded al TN 60 segundos después de recibido el primer fragmento del paquete.

6.5.4 Resultado obtenido

TN receive ICMPv6 error message from NUT
El NUT no debería mandar un mensaje Time Exceeded si el source address es unespecified (::), sin embargo este es enviado.

6.5.5 Solución

Ver test 5.21.

6.6 Test 5.21 - Part D: Echo Request with Unknown Option in Destination Options (Routers and Hosts) Error Condition With Non-Unique Source – Unspecified

6.6.1 Propósito

Verificar que un nodo maneja apropiadamente la recepción de una condición de error causada por un paquete con un Source address que no identifica a un nodo único.

6.6.2 Procedimiento

1. Se envía un paquete tal que:
   - Source address = Unespecified address
   - Destination address = Global address del NUT
   - Option = 135 (Desconocida ya que msb = 10)

6.6.3 Resultados esperados

2. El NUT no debe enviar Parameter Problem Error Message.

6.6.4 Resultado obtenido

TN receive ICMPv6 error message from NUT

El NUT no debería mandar un mensaje Parameter Problem si el source address es unespecified (::), pero este fue enviado.

6.6.5 Solución

Para resolver estos tests se modificó la clase:

```
ip4jvm.net.protocols.extras.ipv6ExtraHeadPrcsrs.IPv6ExtraHeaderProcess
er
```

Método:

```
sendICMP(ICMPv6Header, NetConfigs, Inet6Address)
```

Agregando la siguiente condición:

```
if (!dst.isMulticastAddress() && !IPv6Utils.isUnspecifiedAddress(dst))
```

Esto impide que se envíen paquetes ICMP a direcciones Multicast y Unespecified

7 Errores en segunda corrida

Cuando se agregaron las capacidades de MIPv6 surgieron nuevos errores que fueron resueltos para volver a tener un 100% de Conformance. Algunos de ellos fueron identificados y luego resueltos por el propio grupo que se encontraba realizando estas modificaciones al código, mientras que en otros se ayudó al equipo en cuestión brindando una solución.

8 Tests 3.30 y 3.31

El siguiente código:

```
// icomesana: cambia la manera de agregar una IP por stateless ya que
creemos que así cumple mejor con la rfc4862

NetworkFactory.getNetwork().addIPToInterface(jth, autoIP, ipTentative,
prefixInfOpt.getPrefixLength(), prefixInfOpt.getValidLifeTime(),
prefixInfOpt.getPreferredLifeTime(), doDad,
ipRespondsNeighborSolicitations);
```

hacía fallar los tests 3.30 y 3.31. Por lo tanto fue cambiado a su estado original, a la espera de una solución definitiva.

```
netParams.addAutoIP(jth, autoIP, prefixInfOpt.getValidLifeTime(),
prefixInfOpt.getPreferredLifeTime());
```

9 Referencias

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

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

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

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

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

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

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

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

[9] Open Suse – http://opensuse.org/

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

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

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

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

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

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

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

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

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

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

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

10 Glosario

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

HUT (Host Under Test): Host bajo prueba. En el contexto de este proyecto, un nodo corriendo IP4JVM.

LLA (Link Local Address): Es un tipo de dirección IPv6 utilizado para comunicaciones en una red local o en conexiones punto a punto. El prefijo utilizado es fe80::/64

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

NA (Neighbor Advertisement): Mensaje enviado en respuesta a un NS.

NCE (Neighbor Cache Entry) : Entrada en el Neighbor Cache. El Neighbor Cache es una tabla que contiene los datos sobre todos los nodos vecinos.

NS (Neighbor Solicitation): Es un mensaje para averiguar la dirección IP de un vecino.

RA (Router Advertisement): Mensaje enviado por un router para promocionar su dirección de red.

RS (Router Solicitation): Mensaje enviado por un nodo para descubrir routers en su red.

TLLA (Target Link-layer Address) : La dirección LLA de un nodo destino (Target)

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