https://doi.org/10.25100/iyc.v19i1.2141 TELECOMMUNICATIONS ENGINEERING

NS3-based training system for learning RIPng for IPv6

INGENIERÍA DE TELECOMUNICACIONES

Sistema didáctico basado en NS-3 para el aprendizaje de RIPng para IPv6

Oscar Polanco Sarmiento*

**Escuela de Ingeniería Eléctrica y Electrónica, Universidad del Valle. Cali, Colombia.*
oscar.polanco@correounivalle.edu.co

(Recibido: Febrero 02 de 2016 - Aceptado: Abril 28 de 2016)

Abstract
This article shows the implementation of simulation software which aims to facilitate learning the RIPng protocol (for IPv6). This is based on an IPv6 network with loop topology, on which it is possible to program the tear down moment for one of the least-cost links; said link has been previously used in the routing table calculations which were operating before the tear down. Simulation includes three options which allow the user to choose the control strategy which helps avoiding routing loops: PoisonReverse, SplitHorizon and NoSplitHorizon. The software was developed under the programming environment of the NS-3 network simulator.

Keywords: *ICMPv6, NS-3 Network Simulator, RIPng for IPv6, Split Horizon strategy.*

Resumen
En este artículo, se presenta la implementación de un programa de simulación mediante el cual se busca facilitar el aprendizaje del protocolo RIPng (para IPv6). Este se basa en una red IPv6 con topología en bucle, sobre la cual, se puede programar el instante de falla de uno de los enlaces de menor costo; dicho enlace a su vez ha sido utilizado previamente en el cálculo de la tabla de enrutamiento que se encontraba en operación antes de la falla. La simulación incluye tres opciones que le permiten al usuario seleccionar la estrategia de control que ayuda a evitar los bucles de enrutamiento: PoisonReverse, SplitHorizon y NoSplitHorizon. El programa fue desarrollado bajo el entorno de programación del simulador de red NS-3.

Palabras clave: *Estrategia Split Horizon, ICMPv6, RIPng para IPv6, Simulador de red NS-3.*

1. Introduction

The progress in adopting the IPv6 protocol (Deering & Hinden, 1998) on the side of the different global Internet providers can be evidenced in the measurements the ISOC (Internet Society, 2014) performs periodically in order to show the different dimensions which can be considered in regards to such aspect. As a consequence of this, the topics related to IPv6 have begun to be dealt with in some textbooks which approach content related to TCP/IP architecture (Stevens & Fall, 2011) and the Internetworking discipline (Comer, 2013).

The latter demands the need to approach the topics related to IPv6 in a classroom setting, specifically in those classes in which TCP/IP architecture is the cornerstone. In order to gain a higher practical experience on certain topics, some books (Peterson & Davie, 2011) suggest complementing certain fixed topics with an experiment handbook on network simulation (Aboelela, 2011), which makes use of the Opnet Modeler tool (Opnet, 2012). Nevertheless, no related simulations have yet been approached with some of the fundamental concepts of IPv6 or its related routing protocol.

In 2006 the NS-3 network simulator project was born, which has GNU GPLv2 license terms with architecture and features which are documented on their Website (NS-3, 2006). One of the main characteristics of the NS-3 design is that it improves the model's realism (Riley & Henderson, 2010). Furthermore, it allows users to develop their simulations by using *C++ main()* or Python. In NS-3, simulation results can be obtained via pcap (packet capture) format files, such format is deciphered with *tcpdump* and *Wireshark* (Combs, 1998) in order to represent the captured packets on a physical network, which becomes useful in the network protocol learning drawn from a simulated model.

In 2008, the IPv6 basic implementation was developed for NS-3 (Vincent et al., 2008), which provides support to the following protocols: NDP or Neighbor Discovery Protocol for IP version 6; ICMPv6; and SLAAC or StateLess Address AutoConfiguration (Tomson et al. 2007). On June 2014, the 3.20 NS-3 version supported a dynamic routing protocol for IPv6 for the first time; the supported protocol for IPv6 is RIPng (Malking & Minnear, 1997).

Given that NS-3 allows experimenting with different network protocols in general, and particularly with the IPv6 related protocols, in this article a *C++ main()* program was developed by implementing the libraries for IPv6 on NS-3 version 3.24. This program aims to make a contribution to the practical teaching of the RIPng for IPv6, facilitating the student-protocol interaction. When using it, four functions can be invoked named *linkdown, splitHorizonStrategy, showPings and printRoutingTables*, with which is sought to achieve the following goals:

linkdown. Once simulation has begun and after the routers have consistently learnt the routes to reach the IPv6 destination networks, that is to say, after these have converged, it is possible to tear down a link which is currently transporting IPv6 traffic, right at the moment defined by *linkdown*. With such tear down it is possible to force the routers to recalculate alternate routes and interchange RIPng protocol messages. The above makes possible the capturing and analysis of the RIPng related packets.

splitHorizonStrategy. According to the chosen Split Horizon strategy, it is possible to: 1. Make evident the infinite count issue by means of the *NoSplitHorizon* option. 2. Improve the time to stabilize routing by means of the *SplitHorizon* option, thus avoiding node announcement of the learnt routes from its neighbors when returning. 3. Enable a stronger variation of Split Horizon by means of the *PoisonReverse* option, allowing returning nodes to announce the learnt routes from its neighbor, but poisoning them by associating them to a cost of 16 or infinite.

showPings. When a source system sends ICMPv6 echo request messages (type 128) to a final destination system, the *showPings* option allows observing the recognition responses by means of ICMPv6 echo reply messages (type 129). It is also possible that the emitter receives ICMPv6 messages of inaccessible destination (type 1),

ICMPv6 messages of exceeded time (type 3) for having exceeded the limit of transit leaps, or simply there is no response, depending on the network conditions, at a given moment of the simulation.

printRoutingTables. It allows knowing the content of routing tables within intermediate systems (routers) in the following programmable instances: 1 second before the link's tear down, 20 seconds after the link's tear down and 50 seconds after the link's tear down.

2. Methodology

Figure 1 shows the IPv6 network typology which has been modeled by using the *C++ main()* programming language of NS-3. Generally, the network is made up of the following elements:

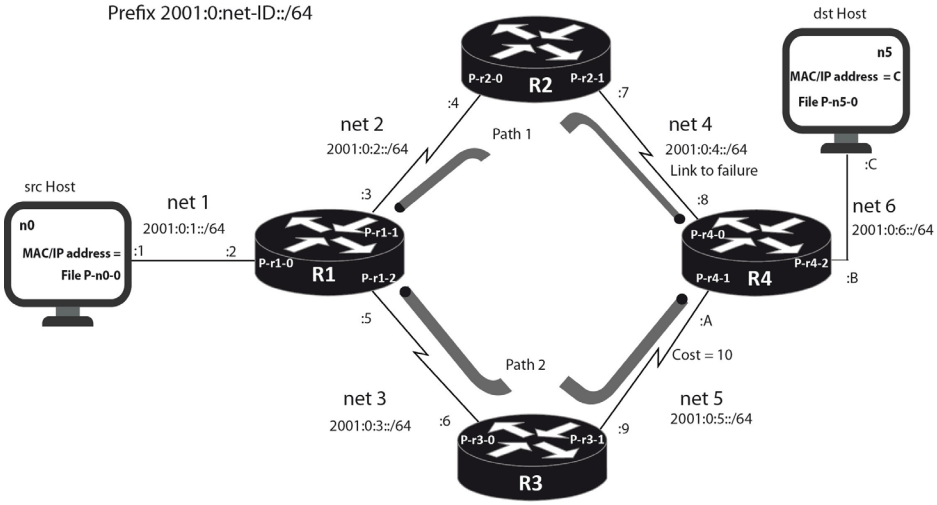

Figure 1. *Network topology model in NS3 with the RIPng protocol for IPv6.*

Six CSMA wired networks (net 1, net 2, net 3, net 4, net 5 and net 6), all networks have a cost of 1 (hop count of 1), except net 5 which has a cost of 10. It has to be said that the only network which fails in the indicated instance by *linkdown* is net 4.

A final emitting system (node n0) which sends ICMPv6 echo request messages towards a final destination system (node n5), which replies every request message with an ICMPv6 echo reply message.

Four intermediate systems or network routers (R1, R2, R3 and R4).

Finally, an IPv6 global routing prefix (2001:0::0/32) assigned to a site, in which 32 bits of subnet are added, value which allows identifying each of the subnets which make up the site, for instance, the prefix 2001:0:3:0::0/64 corresponds to subnet 3 (net 3).

Each node network interface has an assigned MAC address of which hexadecimal value in the first 44 bits is 00:00:00:00:00:0, whereas the value of the last 4 bits are assigned from 1 up to C depending on the order that these are created by the program in NS-3. In regards to the IPv6 addresses, each interface has in the last 64 bits a unique value (*Interface ID*), which allows differentiating it from the rest. Given that NS-3 is operating with Extended Unique Identifier EUI-64™, the value of the last 24 bits of IPv6 and MAC addresses must coincide. Taking into account the latter, in the ends of each link on Figure 1 the last 4 bits of the MAC address are shown and at the same time the last 16 bits of the *Interface ID* are indicated; the 200:00ff:fe00 common value is omitted from the first 48 *Interface ID* bits, since such value repeats itself in every IPv6 address.

Before the net 4 network tear down, communication between nodes n0 and n5 uses path 1 in both the

outbound and returning trips. After the net 4 network tear down and before the necessary time has passed for the routers to converge by means of the RIPng protocol, communication between nodes n0 and n5 is not possible, and as a consequence node 0 is going to receive ICMPv6 error report messages. Furthermore, during this time period routing loops can arise, depending on the Split Horizon strategy that is used for the simulation. Once the routers have converged, the communication between nodes n0 and n5 is reestablished through path 2, which is used in both outbound and returning trips.

When executing the NS-3 simulation, the program generates 12 outbound files in pcap format, which are later on analyzed with *Wireshark*. Figure 1 indicates the name of the associated file for each node interface, for example, the n0 emitting system has the P-n0-0 file assigned to its interface (this nomenclature is used for representing: P = pcap file; n0 = nodo n0; 0 = interface 0), while the R1 system has the three P-r1-0, P-r1-1 and P-r1-2 assigned, one for each interface.

Figure 2 presents the program's flow diagram where the developed system blocks are shown. In block 2 are shown the command names of the four functions mentioned in the introduction section, these names can be used individually or combined. Block 3 shows the command line which the user has to insert for running the program; this uses the *Waf* compiler to run the *ripng-training* program, followed by the function or functions which want to be activated (*linkdown, splitHorizonStrategy, showPings, printRoutingTables*).

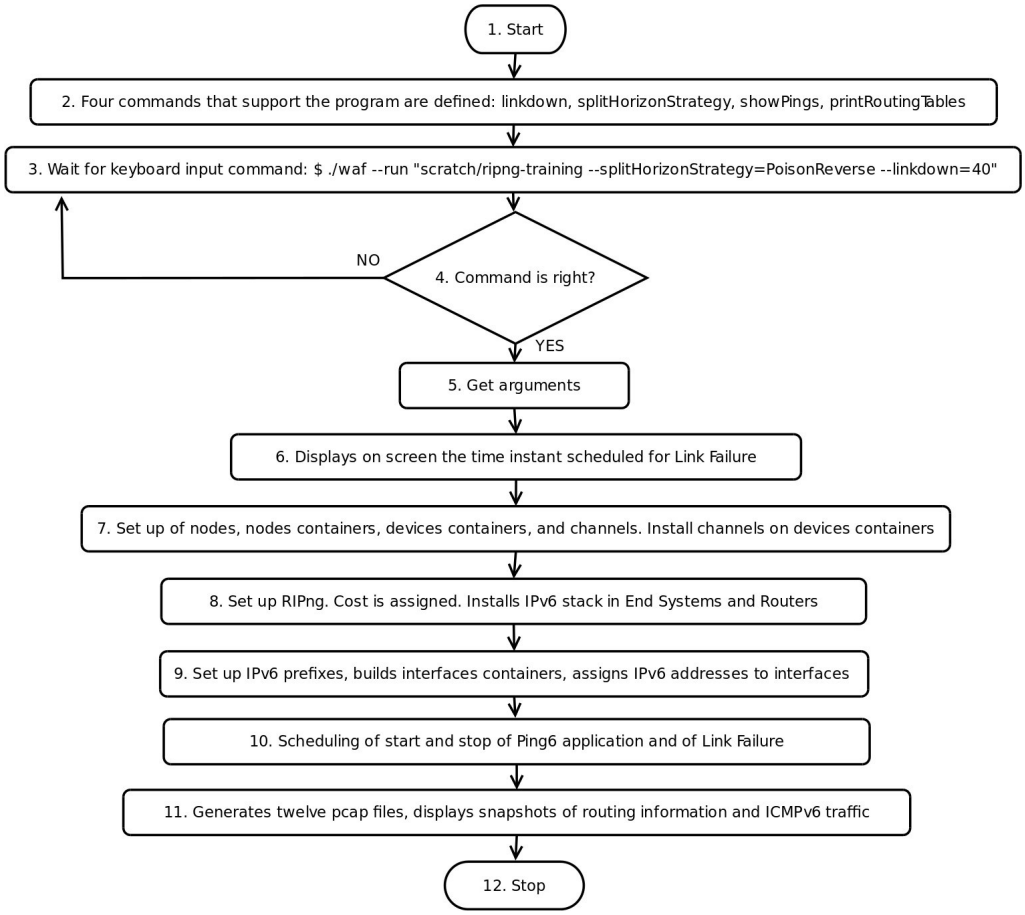

Figure 2. *Program flowchart for learning RIPng.*

Block 4 validates the command introduced by the user according to the functions which want to be run. When the command is correct, block 5 captures, within the respective variables, the required functions; such variables are used to later run the required functions. The simulation has a total running time of 120 seconds and with the *linkdown* command the value is programmed, in seconds, in regards to the instant when the net 4 link tear down should arise; such value can be in the range of 10 to 60 seconds. When values outside of the range are introduced, the program is run, but it uses the values established by default, that is of 10 seconds for values under the lower limit and of 60 seconds for values over the upper limit. By means of block 6 the user is given on-screen feedback by indicating the used value for the moment of the net 4 network tear down.

Block 7 creates the nodes, the node containers, the network device containers and the channels. It also installs the channels to the network device containers. Lines 1 to 10, in Figure 3, show a fraction of the code which creates the n0 and R1 nodes and installs the CSMA channel in the net 1 network.

```
1   Ptr<Node> src = CreateObject<Node> ();
2   Names::Add (“n0”, src);
3   Ptr<Node> a = CreateObject<Node> ();
4   Names::Add (“r1”, a);
5   NodeContainer net1 (src, a);
6   NodeContainer endsystems (src, dst);
7   NodeContainer routers (a, b, c, d);
8   ...
9   CsmaHelper csma;
10  NetDeviceContainer ndc1 = csma.Install (net1);
11  ...
12  RipNgHelper ripNgRouting;
13  ripNgRouting.SetInterfaceMetric (c, 2, 10);
14  ripNgRouting.SetInterfaceMetric (d, 2, 10);
15  ...
16  Ipv6ListRoutingHelper listRH;
17  listRH.Add (ripNgRouting, 0);
18  ...
19  InternetStackHelper internetv6;
```

**Continue Figure 3.*

**Continued Figure 3.*

```
20  internetv6.SetIpv4StackInstall (false);
21  internetv6.SetRoutingHelper (listRH);
22  internetv6.Install (routers);
23  ...
24  InternetStackHelper internetv6Nodes;
25  internetv6Nodes.SetIpv4StackInstall (false);
26  Iinternetv6Nodes.Install (endsystems);
27  ...
28  ipv6.SetBase (Ipv6Address (“2001:0:3::”), Ipv6Prefix (64));
29  Ipv6InterfaceContainer iic3 = ipv6.Assign (ndc3);
30  ...
31  Simulator::Schedule (Seconds (int (linkdown)),
32  &TearDownLink, b, d, 2, 1);
33  ...
34  csma.EnablePcapAll (“ripng”, true);
35  routingHelper.PrintRoutingTableAt (Seconds
36  (linkdown+20), a, routingStream);
    ...
    Simulator::Stop (Seconds (120));
```

Figure 3. *Code fragments of the "ripng-training" program, in C ++ main ().*

Block 8 creates the RIPng protocol, assigns a value of 10 to the net 5 link and leaves by default the cost value of 1 for the other links. It installs the IPv6 stack on the final systems and the routers; in the latter the IPv6 stack remains RIPng enabled. This is present on lines 12 to 26 in Figure 3.

Block 9 uses the API *Ipv6AddressHelper* to create IPv6 prefixes for site subnets. Lines 28 and 29 in Figure 3 illustrate the 2001:0:3::/64 prefix creation, which corresponds to the net 3 subnet. Block 10 programs the moment in which the net 4 link must fail, which is illustrated with line 31 in Figure 3.

In block 10, the ping6 application begins once t=1 and stops when t=110, producing a maximum transfer of 100 packets at a rate of one packet per second. Block 11 generates 12 pcap files by using line 33 in Figure 3. It also records the information from the R1, R2, R3 and R4 routing tables on the *linkdown*-1, *linkdown*+20 and *linkdown*+50 moments. Line 34 allows registering the R1 routing table at the *linkdown*+20 moment.

3. Results

3.1 Routing tables according to the split horizon strategy

Line 37 in Figure 4 allows running the simulation by using the inverse poisoning option, force a tear down on the net 4 link at 40 seconds and activate the option to show the routing tables in-screen for the R1 (node 1), R2 (node 2), R3 (node 3) and R4 (node 4) systems at different instances during the simulation. Lines 39 to 46 show the interesting part in the R2 routing table for moment t=39, just 1 second before tear down happens; in such case, it can be observed that the R2 system knows all the IPv6 networks, including the 2001:0:4::/64 network, soon to fail. Lines 47 to 49 signify that at moment t=60, the 2001:0:4::/64 prefix is no longer in the R2 routing table because the net 4 network has failed; the prefix has been crossed out to illustrate that it does not appear in the table. Lines 50 to 52 show that at moment t=90, the 2001:0:4::/64 prefix is present in the R2 routing table, but with a metric of 16, that is to say, the R1 router has announced a route back to the R2 router about the existence of the 2001:0:4::/64 prefix, but with an infinite cost; this is so R2 does not use R1 when it needs to send IPv6 datagrams towards said prefix as destination.

37	$ ./waf --run "scratch/ripng-training --splitHorizonStrategy=PoisonReverse --printRoutingTables=1 --linkdown=40"
38	...
39	**Node: 2 Time: 39s Ipv6ListRouting table Priority: 0 Protocol: ns3::RipNg**
40	Destination Next Hop Flag Met Ref Use If
41	2001:0:1::/64 fe80::200:ff:fe00:3 UG 2 - - 1
42	2001:0:3::/64 fe80::200:ff:fe00:3 UG 2 - - 1
43	2001:0:6::/64 fe80::200:ff:fe00:8 UG 2 - - 2
44	2001:0:5::/64 fe80::200:ff:fe00:8 UG 2 - - 2
45	2001:0:2::/64 :: U 1 - - 1
46	2001:0:4::/64 :: U 1 - - 2
47	**Node: 2 Time: 60s Ipv6ListRouting table**
48	Destination Next Hop Flag Met Ref Use If
49	~~2001:0:4::/64~~ :: U 1 - - 2 (Lost entry)
50	**Node: 2 Time: 90s Ipv6ListRouting table**

**Continue Figure 4.*

**Continued Figure 4.*

51	Destination Next Hop Flag Met Ref Use If
52	2001:0:4::/64 fe80::200:ff:fe00:3 UG 16 - - 1 (Poisoned entry)
53	...
54	$ ./waf --run "scratch/ripng-training --splitHorizonStrategy=SplitHorizon --linkdown=40"
55	**Node: 2 Time: 90s Ipv6ListRouting table**
56	Destination Next Hop Flag Met Ref Use If
57	~~2001:0:4::/64~~ :: U 1 - - 2 (Lost entry)
58	...
59	$ ./waf --run "scratch/ripng-training --splitHorizonStrategy=NoSplitHorizon --linkdown=40"
60	**Node: 2 Time: 60s Ipv6ListRouting table**
61	Destination Next Hop Flag Met Ref Use If
62	2001:0:4::/64 fe80::200:ff:fe00:3 UG 9 - - 1 (Loop entry in R2)
63	...
64	**Node: 1 Time: 60s Ipv6ListRouting table**
65	Destination Next Hop Flag Met Ref Use If
66	2001:0:4::/64 fe80::200:ff:fe00:4 UG 8 - - 2 (Loop entry in R1)

Figure 4. *Routing tables for different Split Horizon strategies.*

Lines 54 to 57 signify that when the used strategy is Split Horizon, at moment t=90 the R2 system routing table does not have the entry for the 2001:0:4::/64 prefix (crossed out); that is to say, there is no poisoning for such prefix's route. Lines 59 to 66 show that when the strategy disables the Split Horizon characteristic, at moment t=60 a routing loop arises; this is because R1 aims towards R2 and at the same time R2 aims towards R1 in order to reach the 2001:0:4::/64 destination. Although it is not shown in Figure 4, the latter loop also arises to reach the 2001:0:6::/64 prefix.

Figure 5 solely represents some of the captured and stored packets within the pcap file named P-r2-0. Line 68 is a heading which indicates that the following packets correspond to the enabled inverse poisoning strategy. Line 69 indicates that the RIPng packet number 98, present at moment t=43.02, is sent by R1 to R2 in order to indicate to it that it lost the entries to reach the 2001:0:6::/64 (net 6) and 2001:0:4::/64 (net 4) networks; given that the announced cost is infinite (16). In line 70, R1 send

packet 104 to R2 at moment t=87.99 to inform that it knows how to reach all of the networks, except net 4, via a new route. In line 71, R2 sends packet 106 to R1 at moment t=92.96, which poisons the routes announced by R1 on its way back. For the Split Horizon strategy, line 73 indicates that R2 sends packet 102 to R1 at moment t=92.96, which does not poison the routes R1 previously announced on its way back; it only announces net 4 as having infinite cost. For the strategy in which Split Horizon is disabled, line 75 indicates that R1 sends back routes to R2 which it had previously learnt from R2; net 4 with a cost of 2 and net 6 with a cost of 3. Line 76 indicates that R2 sends routes back to R1 which it had previously learnt from R1; net 4 with a cost of 3 and net 6 with a cost of 4. Lines 75 along with 76 indicate that when Split Horizon is disabled, routing loops can be created.

67	No	Time	Source	Destination	Info
68					**Poison Reverse**
69	98	43.02	fe80::200:ff:fe00:3 (R1)	ff02::9 (R2)	2001:0:5::/64 (2) 2001:0:6::/64 (16) 2001:0:4::/64 (16) 2001:0:1::/64 (1) 2001:0:2::/64 (16)
70	104	87.99	fe80::200:ff:fe00:3 (R1)	ff02::9 (R2)	2001:0:5::/64 (2) 2001:0:6::/64 (12) 2001:0:4::/64 (16) 2001:0:1::/64 (1) 2001:0:2::/64 (16) 2001:0:3::/64 (1)
71	106	92.96	fe80::200:ff:fe00:4 (R2)	ff02::9 (R1)	2001:0:1::/64 (16) 2001:0:3::/64 (16) 2001:0:6::/64 (16) 2001:0:5::/64 (16) 2001:0:2::/64 (16) 2001:0:4::/64 (16)
72					**Split Horizon**
73	102	92.96	fe80::200:ff:fe00:4 (R2)	ff02::9 (R1)	2001:0:4::/64 (16)
74					**No Split Horizon**
75	98	43.02	fe80::200:ff:fe00:3 (R1)	ff02::9 (R2)	2001:0:5::/64 (2) 2001:0:6::/64 (3) 2001:0:4::/64 (2) 2001:0:1::/64 (1) 2001:0:2::/64 (1) 2001:0:3::/64 (1)
76	99	43.55	fe80::200:ff:fe00:4 (R2)	ff02::9 (R1)	2001:0:6::/64 (4) 2001:0:5::/64 (3) 2001:0:4::/64 (3)

Figure 5. *Traces from Wireshark for RIPng with different Split Horizon strategies (P-r2-0).*

3.2 ICMPv6 control and error report message

Lines 77 to 81 in Figure 6 represent some ICMPv6 messages from the P-r2-0 file. Line 79 is a Destination unreachable ICMPv6 message (type 1) which R2 sends to the n0 node. Line 80 is a Redirect ICMPv6 message (type 137) which R2 sends R1. Finally, line 81 is a Time exceeded ICMPv6 message (type 3) which R2 sends to the n0 node.

77	Time	Source	Destination	Info
78				**No Split Horizon**
79	40:00	2001:0:2:0:200:ff:fe00:4	2001:0:1:0:200:ff:fe00:1	unreachable route 2001:0:6:0:200:ff:fe00:c, ICMPv6 type 1
80	44:00	Mac: 00:00:00:00:00:04 fe80::200:ff:fe00:4	Mac: 00:00:00:00:00:03 2001:0:1:0:200:ff:fe00:1	redirect, 2001:0:6:0:200:ff:fe00:c to fe80::200:ff:fe00:3, ICMPv6 type 137
81	44.37	2001:0:2:0:200:ff:fe00:4	2001:0:1:0:200:ff:fe00:1	time exceeded in-transit for 2001:0:6:0:200:ff:fe00:c, ICMPv6 type 3

Figure 6. *Traces for ICMPv6 messages, No Split Horizon strategy.*

3.3 Contribution and justification

Running the software generates information which allows students to contrast theoretical concepts of RIPng protocol to the true behavior of said protocol within a simulated network. The software incorporates parameters which allow controlling various network events and defining the information

that is required to be obtained in a specific manner. The software is flexible since it allows different types of changes, as for example within network topology and IPv6 addresses which are desired to be assigned to the nodes.

4. Discussion

When the RIPng protocol is enabled within the intermediate systems, it was possible to verify that the routers send the routing tables periodically every 30 seconds independently from the used Split Horizon strategy, and they also send the tables when there is a specific event, as usually is the reestablishment or tear down of a link. It was also evidenced that RIPng uses a 1 to 16 metric, which represents the hop count and also shows 15 as the maximum amount of hops. When the *linkdown* function is running, it is observed that, as a matter of fact, it is possible to control the desired moment to tear down the net 4 link (by simulating a failure), and thus force the intermediate IPv6 network systems to recalculate the routing tables which allow to obtain new routes which take into account the other available links. When trying to revert the *linkdown* function with the purpose of uploading the net 4 link after having closed it, it was found that even though the net 4 link uploads, the R2 (2001:0:4:0:200:ff:fe00:7) and R4 (2001:0:4:0:200:ff:fe00:8) interfaces, which are common to the net 4 link, only recover the local-link address scope (fe80::200:ff:fe00:7 and fe80::200:ff:fe00:8) and lose their global address, thus the 2001:0:4::/64 disappears from the routing tables; this suggests that a problem-to-solve has been identified within the implementation of the IPv6 protocol in NS-3. In terms of routing loop prevention, when contrasting the simulations used by the Poison Reverse strategies, Split Horizon and disabled Split Horizon, it is found that the most helpful one for said purpose is inverse poisoning, the intermediate one is Split Horizon and the worst is the one which disables Split Horizon. In the reverse poisoning strategy, during the first 120 seconds of simulation, 106 packets were captured in the P-r2-0, and in the last packet it is evidenced that R2 sends R1 the prefixes which it previously learnt from R1, but it sends them at an infinite cost, that is to say, it sends the poisoned routes. In the Split Horizon strategy, during the same 120 seconds of simulation, 102 packets were captured in the P-r2-0 file, and it is evidenced that in the last packet, R2 does not poison nor sends R1 the prefixes previously learnt from R1, it only sends the announcement of net 4 at infinite cost. Finally, in the strategy which disables Split Horizon, during the same 120 seconds of simulation, 2,971 packets were captured in the P-r2-0 file, and it is evidenced that after moments t=43.02 and t=43.55, along packets 98 and 99, the R1 and R2 systems initiate an interchange of route announcements which cause the setting up of the routing loops. It is also important to highlight that the elevated packet number is largely due to the fact that the IPv6 datagrams with destination to the n5 node keep looping between R1 and R2, in addition to the time exceeded ICMPv6 messages (type 3) due to the hop count limit and redirect ICMPv6 messages (type 137) which are generated owing to this.

The option for printing routing tables is a powerful tool given that it allows examining the routing tables of any IPv6 network system at any time. Specifically, at the moments of interest for this case: one second before the tear down, 20 seconds after the tear down and 50 seconds after the tear down. With the obtained output saved on a text file, it was possible to evince that the R1, R2, R3 and R4 systems had the expected entry routes; it is worthy to note that it was possible to identify those entries that a system had lost, those which had arrived poisoned and those which caused routing loops. Meanwhile, the *showPings* option allows having a general idea about what is going on with the traffic that the n0 node sends to the n5 node.

Finally, when examining the captured packets with *Wireshark*, it is observed that the R1 and R2 systems send type 1 and type 3 ICMPv6 messages to the n0 node in order to indicate unreachable destination and exceeded hop count limit, respectively. This indicates a standard IPv6 protocol procedure by the time the router has lost a route or by the time the packet's hop count has reached zero. Notwithstanding, type 137 ICMPv6 messages were also observed, which are route correctors sent by R2 to R1 when routing loops arise; since R2 assumes that R1 has the best route

to reach the 2001:0:6::/64 network. This last part implies an incorrect operation of the NS-3 IPv6 stack inasmuch as the source of the message is not from the R1 system, but instead from the n0 node.

5. Conclusions

With the developed software an IPv6 network can be simulated, made up of nodes which have the IPv6 stack installed and which are interconnected by CSMA link. Within the nodes which operate as routers, a RIPng routing protocol must be additionally installed. In the simulation, the moment of a specific link tear down can be programmed as well as the Split Horizon strategy which is going to be used in the RIPng protocol. Based on the information generated by the simulation, it is possible to analyze the node routing tables, the network convergence times and the finest traffic details of the interchanged packets. All of the above allows verifying the grounded concepts in regards to RIPng protocol.

6. References

Aboelela, E. (2011). *Network Simulation Experiments Manual*. (5th Edition). Burlington, Massachusetts: Morgan Kaufmann.

Combs, G. (1998). *Wireshark*. [Viewed in December 2015] http://www.wireshark.org

Comer, D.E. (2013). *Internetworking with TCP/IP Volume One*. (6th Edition). New Jersey: Addison-Wesley.

Deering, S. & Hinden, R. (1998). *Internet Protocol, Version 6 (IPv6) Specification*. In IETF (The Internet Engineering Task Force) Request for Comments 2460. [Viewed in December 2015] http://www.ietf.org/rfc/rfc2460.txt

Internet Society. (2014). *The future is forever, world IPv6 launch – Meachurements*. [Viewed in December 2015] http://www.worldipv6launch.org/measurements/

Malking, G. & Minnear, R. (1997). *RIPng for IPv6*. In IETF (The Internet Engineering Task Force) Request for Comments 2080. [Viewed in December 2015] https://tools.ietf.org/html/rfc2080

Ns-3 (2006). *The ns-3 Project*. [Viewed in December 2015] http://www.nsnam.org/

Opnet Technologies. (2012). *Application and network performance*. [Viewed in December 2015] http://www.opnet.com/

Peterson, L.L. & Davie, B.S. (2011). *Computer Networks: A Systems Approach*. (Fifth Edition). Burlington, Massachusetts: Morgan Kaufmann.

Riley, G.F. & Henderson T.R. (2010). *Modeling and Tools for Network Simulation*. Berlin Heidelberg: Springer-Verlag.

Stevens, W.R. & Fall, K.R. (2011). *TCP/IP Illustrated, Volume 1: The Protocols*. (2nd Edition). New Jersey: Addison-Wesley Professional.

Thomson, S., Narten, T. & Jinmei, T. (2007). *IPv6 Stateless Address Autoconfiguration*, Internet Engineering Task Force Request for Comments (RFC) 4862. [Viewed in December 2015] http://www.ietf.org/rfc/rfc4862.txt

Vincent, S., Montavont, J. & Montavont, N. (2008). *Implementation of an IPv6 stack for ns-3*, Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools. Athens, Greece.