Simulating RIPng Protocol Scenarios Using NS-3

Sourabh S Shenoy

August 26, 2014

Abstract

Routing Information Protocol next Generation (RIPng) is one of the protocols in IPv6 that allows routers to dynamically communicate with each other. RIPng is similar to RIP in IPv4. With the need to shift from IPv4 to IPv6 being clearly evident, the question of having advanced routing protocols that comply with IPv6 also comes into picture. RIPng is one of the protocols that has support for IPv6 networking. It is a Distance Vector Routing Protocol. Keeping this in mind, some real-time scenarios are implemented to simulate the working of RIPng.

Contents

1 INTRODUCTION:

1.1 Routing Information Protocol next gen (RIPng)

The Routing Information Protocol next generation (RIPng) is a distance-vector routing protocol, which is based on the Bellman-Ford Algorithm. It employs the hop count as a routing metric. The protocol is limited to networks whose diameter is 15 hops. A hop count of 16 is considered an infinite distance. In other words, the route is considered unreachable. RIPng implements the split horizon, route poisoning and hold down mechanisms to prevent incorrect routing information from being propagated.

Any router that uses RIPng is assumed to have interfaces to one or more networks. These are its directly-connected networks. The protocol relies on certain information about these networks, the most important of which is its metric which is an integer between 1 and 15. In addition to the metric, each network will have an IPv6 destination address prefix and prefix length associated with it. Each router that implements RIPng is assumed to have a routing table. This table has one entry for every destination that is reachable.

Each entry contains at least the following information:

- The IPv6 prefix of the destination.
- A metric, which represents the total cost of getting a datagram from the router to that destination.
- The IPv6 address of the next router along the path to the destination.
- A flag to indicate that information about the route has changed recently. This will be referred to as the ”route change flag.”

1.2 Network Simulator-3

Network Simulator-3 (NS-3) is an open source discrete-event network simulator targeted for Research and Education.

2 MOTIVATION:

As the need to shift from IPv4 to IPv6 arises, we need to use protocols that support IPv6 networking. RIPng protocol works with 128 bit IP addresses. It is important to understand how RIPng works. Following are some scenarios that simulate the working of RIPng using NS-3.

3 SIMULATION SCENARIOS:

Lines beginning with '//' are comments

3.1 RIPng Scenario-1

```
// Network topology
//
//     SRC
//      | <=== Source Network
//      A------B
//      |      |
//      C------D
//             | <=== Destination Network
//            DST
//

// Cost of different networks is as follows:
// A to B : 5, B to D : 3, A to C : 2, C to D : 4
// A, B, C and D are RIPng routers.
// A and D are configured with static addresses.
// SRC and DST will exchange packets.
//
// After about 3 seconds, the topology is built, and
   Echo Reply will be received.
// After 30 seconds, the link between A and C will
   break, causing a route failure.
// After 31 seconds from the failure, the routers
   will recover from the failure.
// If "showPings" is enabled, the user will see:
// 1) if the ping has been acknowledged
// 2) if a Destination Unreachable has been received
   by the sender
// 3) nothing, when the Echo Request has been
   received by the destination but
//    the Echo Reply is unable to reach the sender.
// Examining the .pcap files with Wireshark can
   confirm this effect.

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv6-static-routing-helper.h"
#include "ns3/ipv6-routing-table-entry.h"

using namespace ns3;
//Define a logging component for this file which can
   later be enabled or disabled with the NS_LOG
   environment variable
NS_LOG_COMPONENT_DEFINE ("RipNg-1");

//To tear down the link between any two interfaces
void TearDownLink (Ptr<Node> nodeA, Ptr<Node> nodeB,
   uint32_t interfaceA, uint32_t interfaceB)
{
  nodeA->GetObject<Ipv6> ()->SetDown (interfaceA);
  nodeB->GetObject<Ipv6> ()->SetDown (interfaceB);
}

int main (int argc, char **argv)
```

```
{
//Declare a verbose flag that can be set to true
   using the command line to enable log components
  bool verbose = false;
//Split horizon with Poison Reverse is used
  std::string SplitHorizon ("PoisonReverse");
  bool showPings = false;

//Creates an instance of CommandLine class to enable
   passsing command line arguments
  CommandLine cmd;
  cmd.AddValue ("verbose", "turn on log components",
     verbose);
  cmd.AddValue ("showPings", "Show Ping6 reception",
     showPings);
  cmd.AddValue ("splitHorizonStrategy", "Split
     Horizon strategy to use (NoSplitHorizon,
     SplitHorizon, PoisonReverse)", SplitHorizon);
//To parse the command line arguments
  cmd.Parse (argc, argv);

//Enables the Logging Components of all the levels
   above the level specified if verbose is set to true
  if (verbose)
    {
      LogComponentEnable ("RipNgSimpleRouting",
         LOG_LEVEL_INFO);
      LogComponentEnable ("RipNg", LOG_LEVEL_ALL);
      LogComponentEnable ("Icmpv6L4Protocol",
         LOG_LEVEL_INFO);
      LogComponentEnable ("Ipv6Interface",
        LOG_LEVEL_ALL);
      LogComponentEnable ("Icmpv6L4Protocol",
        LOG_LEVEL_ALL);
      LogComponentEnable ("NdiscCache", LOG_LEVEL_ALL
        );
      LogComponentEnable ("Ping6Application",
        LOG_LEVEL_ALL);
    }

  if (showPings)
    {
      LogComponentEnable ("Ping6Application",
        LOG_LEVEL_INFO);
    }

//Enables either Split Horizon, No Split horizon or
  poison Reverse depending on what is selected
  if (SplitHorizon == "NoSplitHorizon")
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
          EnumValue (RipNg::NO_SPLIT_HORIZON));
    }
  else if (SplitHorizon == "SplitHorizon")
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
          EnumValue (RipNg::SPLIT_HORIZON));
    }
  else
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::POISON_REVERSE));
    }

//To create the Nodes
  NS_LOG_INFO ("Creating nodes.");
  Ptr<Node> src = CreateObject<Node> ();
//Associates the node with the name provided in the
   first argument
  Names::Add ("SrcNode", src);
  Ptr<Node> dst = CreateObject<Node> ();
  Names::Add ("DstNode", dst);
  Ptr<Node> a = CreateObject<Node> ();
  Names::Add ("RouterA", a);
  Ptr<Node> b = CreateObject<Node> ();
  Names::Add ("RouterB", b);
  Ptr<Node> c = CreateObject<Node> ();
  Names::Add ("RouterC", c);
  Ptr<Node> d = CreateObject<Node> ();
  Names::Add ("RouterD", d);

//Defines a NodeContainer object that holds Devices
  NodeContainer net1 (src, a);
  NodeContainer net2 (a, b);
  NodeContainer net3 (a, c);
  NodeContainer net4 (b, d);
  NodeContainer net5 (c, d);
  NodeContainer net6 (d, dst);
  NodeContainer routers (a, b, c, d);
  NodeContainer nodes (src, dst);
```

```
//Creating CSMA channels
  NS_LOG_INFO ("Creating channels.");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue
      (5000000));
  csma.SetChannelAttribute ("Delay", TimeValue (
     MilliSeconds (2)));

//Install CSMA Between the nodes
  NetDeviceContainer ndc1 = csma.Install (net1);
  NetDeviceContainer ndc2 = csma.Install (net2);
  NetDeviceContainer ndc3 = csma.Install (net3);
  NetDeviceContainer ndc4 = csma.Install (net4);
  NetDeviceContainer ndc5 = csma.Install (net5);
  NetDeviceContainer ndc6 = csma.Install (net6);

  NS_LOG_INFO ("Configuring RIPng on the Routers");
//Defines an RipNgHelper object that helps configure
  RipNg
  RipNgHelper ripNgRouting;

//To exclude RIPng at the specified interface
//Interfaces start from 0 and are added sequentially
//However Interface 0 is for loopback address
  ripNgRouting.ExcludeInterface (a, 1);
  ripNgRouting.ExcludeInterface (d, 3);

//Sets the interface metric for the specified links
  ripNgRouting.SetInterfaceMetric (a, 2, 5);
  ripNgRouting.SetInterfaceMetric (a, 3, 2);
  ripNgRouting.SetInterfaceMetric (b, 1, 5);
  ripNgRouting.SetInterfaceMetric (b, 2, 3);
  ripNgRouting.SetInterfaceMetric (c, 1, 2);
  ripNgRouting.SetInterfaceMetric (c, 2, 4);
  ripNgRouting.SetInterfaceMetric (d, 1, 3);
  ripNgRouting.SetInterfaceMetric (d, 2, 4);

  Ipv6ListRoutingHelper listRH;
  listRH.Add (ripNgRouting, 0);

  InternetStackHelper internetv6;
  internetv6.SetIpv4StackInstall (false);
  internetv6.SetRoutingHelper (listRH);
  internetv6.Install (routers);

  InternetStackHelper internetv6Nodes;
  internetv6Nodes.SetIpv4StackInstall (false);
  internetv6Nodes.Install (nodes);

//Assigning IPv6 addresses
//Source and Destination have Global Addresses
//Rest of the routers need only link-local addresses
   for routing within the network
  NS_LOG_INFO ("Assign IPv6 Addresses.");
  Ipv6AddressHelper ipv6;

  ipv6.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix
     (64));
  Ipv6InterfaceContainer iic1 = ipv6.Assign (ndc1);
  iic1.SetForwarding (1, true);
  iic1.SetDefaultRouteInAllNodes (1);
```

```
ipv6.SetBase (Ipv6Address ("2001:0:1::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic2 = ipv6.Assign (ndc2);
iic2.SetForwarding (0, true);
iic2.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:2::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic3 = ipv6.Assign (ndc3);
iic3.SetForwarding (0, true);
iic3.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:3::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic4 = ipv6.Assign (ndc4);
iic4.SetForwarding (0, true);
iic4.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:4::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic5 = ipv6.Assign (ndc5);
iic5.SetForwarding (0, true);
iic5.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:2::"), Ipv6Prefix
  (64));
Ipv6InterfaceContainer iic6 = ipv6.Assign (ndc6);
iic6.SetForwarding (0, true);
iic6.SetDefaultRouteInAllNodes (0);

NS_LOG_INFO ("Create Applications.");
```

```
  uint32_t packetSize = 1024;
  uint32_t maxPacketCount = 100;
  Time interPacketInterval = Seconds (1.0);
  Ping6Helper ping6;

//Set the source and destination address
  ping6.SetLocal (iic1.GetAddress (0, 1));
  ping6.SetRemote (iic6.GetAddress (1, 1));
  ping6.SetAttribute ("MaxPackets", UintegerValue (
     maxPacketCount));
  ping6.SetAttribute ("Interval", TimeValue (
     interPacketInterval));
  ping6.SetAttribute ("PacketSize", UintegerValue (
     packetSize));
  ApplicationContainer apps = ping6.Install (src);
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (110.0));

  AsciiTraceHelper ascii;
//To create the trace file
  csma.EnableAsciiAll (ascii.CreateFileStream ("ripng
     -1.tr"));
//Enables generation of .pcap files that can be
  examined using wireshark
  csma.EnablePcapAll ("ripng-1", true);

//Call the function defined above to tear down the
  link between two nodes
  Simulator::Schedule (Seconds (30), &TearDownLink, a
     , c, 3, 1);
```

```
// Now, do the actual simulation.
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds (120));
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

3.2 RIPng Scenario-2

```
// Network topology
//
//    SRC
//     | <=== Source Network
//     |
//     A------B------C------D
//     |------------|      |
//                         | <=== Destination Network
//                        DST
//
// Cost of different networks is as follows:
// B to C : 5, All other links, Cost is 1
// A, B, C and D are RIPng routers.
// A and D are configured with static addresses.
// SRC and DST will exchange packets.
//
// After about 3 seconds, the topology is built, and
   Echo Reply will be received.
// After 40 seconds, the link between A and C will
   break, causing a route failure.
// After 24 seconds from the failure, the routers
   will recover from the failure.
// Split Horizoning should affect the recovery time,
   but it is not. See the manual
// for an explanation of this effect.
//
// If "showPings" is enabled, the user will see:
// 1) if the ping has been acknowledged
// 2) if a Destination Unreachable has been received
   by the sender
// 3) nothing, when the Echo Request has been
   received by the destination but
//    the Echo Reply is unable to reach the sender.
// Examining the .pcap files with Wireshark can
   confirm this effect.

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv6-static-routing-helper.h"
#include "ns3/ipv6-routing-table-entry.h"

using namespace ns3;
//Define a logging component for this file which can
   later be enabled or disabled with the NS_LOG
   environment variable
NS_LOG_COMPONENT_DEFINE ("RipNg-2");

//To tear down the link between any two interfaces
void TearDownLink (Ptr<Node> nodeA, Ptr<Node> nodeB,
   uint32_t interfaceA, uint32_t interfaceB)
{
  nodeA->GetObject<Ipv6> ()->SetDown (interfaceA);
  nodeB->GetObject<Ipv6> ()->SetDown (interfaceB);
}

int main (int argc, char **argv)
```

```
{
//Declare a verbose flag that can be set to true
   using the command line to enable log components
  bool verbose = false;
  bool printRoutingTables = false;
  bool showPings = false;
//Split horizon with Poison Reverse is used
  std::string SplitHorizon ("PoisonReverse");

//Creates an instance of CommandLine class to enable
   passsing command line arguments
  CommandLine cmd;
  cmd.AddValue ("verbose", "turn on log components",
     verbose);
  cmd.AddValue ("printRoutingTables", "Print routing
     tables at 30, 60 and 90 seconds",
     printRoutingTables);
  cmd.AddValue ("showPings", "Show Ping6 reception",
     showPings);
  cmd.AddValue ("splitHorizonStrategy", "Split
     Horizon strategy to use (NoSplitHorizon,
     SplitHorizon, PoisonReverse)", SplitHorizon);
//To parse the command line arguments
  cmd.Parse (argc, argv);

//Enables the Logging Components of all the levels
   above the level specified if verbose is set to true
  if (verbose)
    {
      LogComponentEnable ("RipNgSimpleRouting",
         LOG_LEVEL_INFO);
      LogComponentEnable ("RipNg", LOG_LEVEL_ALL);
      LogComponentEnable ("Icmpv6L4Protocol",
         LOG_LEVEL_INFO);
      LogComponentEnable ("Ipv6Interface",
         LOG_LEVEL_ALL);
      LogComponentEnable ("Icmpv6L4Protocol",
         LOG_LEVEL_ALL);
      LogComponentEnable ("NdiscCache", LOG_LEVEL_ALL
         );
      LogComponentEnable ("Ping6Application",
         LOG_LEVEL_ALL);
    }

  if (showPings)
    {
      LogComponentEnable ("Ping6Application",
         LOG_LEVEL_INFO);
    }

//Enables either Split Horizon, No Split horizon or
   poison Reverse depending on what is selected
  if (SplitHorizon == "NoSplitHorizon")
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
          EnumValue (RipNg::NO_SPLIT_HORIZON));
    }
  else if (SplitHorizon == "SplitHorizon")
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
          EnumValue (RipNg::SPLIT_HORIZON));
    }
  else
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
          EnumValue (RipNg::POISON_REVERSE));
    }

//To create the Nodes
  NS_LOG_INFO ("Create nodes.");
  Ptr<Node> src = CreateObject<Node> ();
//Associates the node with the name provided in the
   first argument
  Names::Add ("SrcNode", src);
  Ptr<Node> dst = CreateObject<Node> ();
  Names::Add ("DstNode", dst);
  Ptr<Node> a = CreateObject<Node> ();
  Names::Add ("RouterA", a);
  Ptr<Node> b = CreateObject<Node> ();
  Names::Add ("RouterB", b);
  Ptr<Node> c = CreateObject<Node> ();
  Names::Add ("RouterC", c);
  Ptr<Node> d = CreateObject<Node> ();
  Names::Add ("RouterD", d);

//Defines a NodeContainer object that holds Devices
  NodeContainer net1 (src, a);
  NodeContainer net2 (a, b);
  NodeContainer net3 (a, c);
  NodeContainer net4 (b, c);
  NodeContainer net5 (c, d);
  NodeContainer net6 (d, dst);
  NodeContainer routers (a, b, c, d);
  NodeContainer nodes (src, dst);

//Creating CSMA channels
  NS_LOG_INFO ("Create channels.");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue
      (5000000));
  csma.SetChannelAttribute ("Delay", TimeValue (
     MilliSeconds (2)));

//Install CSMA Between the nodes
  NetDeviceContainer ndc1 = csma.Install (net1);
  NetDeviceContainer ndc2 = csma.Install (net2);
  NetDeviceContainer ndc3 = csma.Install (net3);
  NetDeviceContainer ndc4 = csma.Install (net4);
  NetDeviceContainer ndc5 = csma.Install (net5);
  NetDeviceContainer ndc6 = csma.Install (net6);

  NS_LOG_INFO ("Create IPv6 and routing");
//Defines an RipNgHelper object that helps configure
   RipNg
  RipNgHelper ripNgRouting;

//To exclude RIPng at the specified interface
//Interfaces start from 0 and are added sequentially
//However Interface 0 is for loopback address
  ripNgRouting.ExcludeInterface (a, 1);
  ripNgRouting.ExcludeInterface (d, 2);

//Sets the interface metric for the specified links
  ripNgRouting.SetInterfaceMetric (b, 2, 5);
  ripNgRouting.SetInterfaceMetric (c, 2, 5);

  Ipv6ListRoutingHelper listRH;
  listRH.Add (ripNgRouting, 0);

  InternetStackHelper internetv6;
  internetv6.SetIpv4StackInstall (false);
  internetv6.SetRoutingHelper (listRH);
  internetv6.Install (routers);

  InternetStackHelper internetv6Nodes;
  internetv6Nodes.SetIpv4StackInstall (false);
  internetv6Nodes.Install (nodes);

//Assigning IPv6 addresses
//Source and Destination have Global Addresses
//Rest of the routers need only link-local addresses
   for routing within the network

  NS_LOG_INFO ("Assign IPv6 Addresses.");
  Ipv6AddressHelper ipv6;

  ipv6.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix
     (64));
  Ipv6InterfaceContainer iic1 = ipv6.Assign (ndc1);
  iic1.SetForwarding (1, true);
  iic1.SetDefaultRouteInAllNodes (1);

  ipv6.SetBase (Ipv6Address ("2001:0:1::"),
     Ipv6Prefix (64));
  Ipv6InterfaceContainer iic2 = ipv6.Assign (ndc2);
```

```
iic2.SetForwarding (0, true);
iic2.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:2::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic3 = ipv6.Assign (ndc3);
iic3.SetForwarding (0, true);
iic3.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:3::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic4 = ipv6.Assign (ndc4);
iic4.SetForwarding (0, true);
iic4.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:4::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic5 = ipv6.Assign (ndc5);
iic5.SetForwarding (0, true);
iic5.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:2::"), Ipv6Prefix
  (64));
Ipv6InterfaceContainer iic6 = ipv6.Assign (ndc6);
iic6.SetForwarding (0, true);
iic6.SetDefaultRouteInAllNodes (0);

if (printRoutingTables)
  {
    RipNgHelper routingHelper;
    Ptr<OutputStreamWrapper> routingStream = Create
      <OutputStreamWrapper> (&std::cout);

    routingHelper.PrintRoutingTableAt (Seconds
      (30.0), a, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (30.0), b, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (30.0), c, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (30.0), d, routingStream);

    routingHelper.PrintRoutingTableAt (Seconds
      (60.0), a, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (60.0), b, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (60.0), c, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (60.0), d, routingStream);

    routingHelper.PrintRoutingTableAt (Seconds
      (90.0), a, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (90.0), b, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (90.0), c, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (90.0), d, routingStream);
  }
```

```
  NS_LOG_INFO ("Create Applications.");
  uint32_t packetSize = 1024;
  uint32_t maxPacketCount = 100;
  Time interPacketInterval = Seconds (1.0);
  Ping6Helper ping6;

//Set the source and destination address
  ping6.SetLocal (iic1.GetAddress (0, 1));
  ping6.SetRemote (iic6.GetAddress (1, 1));
  ping6.SetAttribute ("MaxPackets", UintegerValue (
     maxPacketCount));
  ping6.SetAttribute ("Interval", TimeValue (
     interPacketInterval));
  ping6.SetAttribute ("PacketSize", UintegerValue (
     packetSize));
  ApplicationContainer apps = ping6.Install (src);
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (110.0));

  AsciiTraceHelper ascii;
//To create the trace file
  csma.EnableAsciiAll (ascii.CreateFileStream ("ripng
     -2.tr"));
//Enables generation of .pcap files that can be
   examined using wireshark
  csma.EnablePcapAll ("ripng-2", true);

//Call the function defined above to tear down the
   link between two nodes
  Simulator::Schedule (Seconds (40), &TearDownLink, a
     , c, 3, 1);
```

```
/* Now, do the actual simulation. */
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds (120));
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

3.3 RIPng Scenario-3

```
// Network topology
//
//     SRC
//       | <=== Source Network
//           A-----B-----C
//              |     /|
//                   |    / |
//              |   /  |
//              |  /   |
//              |/     |
//              D-----E
//                     | <=== Destination Network
//                    DST
//
//
// Cost of different networks is as follows:
// B to C : 5, C to E : 3, All other links cost 1
// A, B, C, D and E are RIPng routers.
// A and E are configured with static addresses.
// SRC and DST will exchange packets.
//
// After about 7 seconds, the topology is built, and
   Echo Reply will be received.
// After 30 seconds, the link between B and D will
   break, causing a route failure.
// After 15 seconds from the failure, the routers
   will recovery from the failure.
// After 30 seconds from recovery, the link between C
    and D will break, causing a route failure.
```

```
// After 34 seconds from the failure, the routers
   will recover again from the failure.
// Split Horizoning should affect the recovery time,
   but it is not. See the manual
// for an explanation of this effect.
//
// If "showPings" is enabled, the user will see:
// 1) if the ping has been acknowledged
// 2) if a Destination Unreachable has been received
   by the sender
// 3) nothing, when the Echo Request has been
   received by the destination but
//    the Echo Reply is unable to reach the sender.
// Examining the .pcap files with Wireshark can
   confirm this effect.

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv6-static-routing-helper.h"
#include "ns3/ipv6-routing-table-entry.h"

using namespace ns3;
//Define a logging component for this file which can
   later be enabled or disabled with the NS_LOG
   environment variable
NS_LOG_COMPONENT_DEFINE ("RipNg-3");

//To tear down the link between any two interfaces
```

```
void TearDownLink (Ptr<Node> nodeA, Ptr<Node> nodeB,
   uint32_t interfaceA, uint32_t interfaceB)
{
  nodeA->GetObject<Ipv6> ()->SetDown (interfaceA);
  nodeB->GetObject<Ipv6> ()->SetDown (interfaceB);
}

int main (int argc, char **argv)
{
//Declare a verbose flag that can be set to true
   using the command line to enable log components
  bool verbose = false;
  bool printRoutingTables = false;
  bool showPings = false;
//Split horizon with Poison Reverse is used
  std::string SplitHorizon ("PoisonReverse");

//Creates an instance of CommandLine class to enable
   passsing command line arguments
  CommandLine cmd;
  cmd.AddValue ("verbose", "turn on log components",
     verbose);
  cmd.AddValue ("printRoutingTables", "Print routing
     tables at 30, 60 and 90 seconds",
     printRoutingTables);
  cmd.AddValue ("showPings", "Show Ping6 reception",
     showPings);
  cmd.AddValue ("splitHorizonStrategy", "Split
     Horizon strategy to use (NoSplitHorizon,
     SplitHorizon, PoisonReverse)", SplitHorizon);
//To parse the command line arguments
```

```
  cmd.Parse (argc, argv);

//Enables the Logging Components of all the levels
   above the level specified if verbose is set to true
  if (verbose)
    {
      LogComponentEnable ("RipNgSimpleRouting",
         LOG_LEVEL_INFO);
      LogComponentEnable ("RipNg", LOG_LEVEL_ALL);
      LogComponentEnable ("Icmpv6L4Protocol",
         LOG_LEVEL_INFO);
      LogComponentEnable ("Ipv6Interface",
         LOG_LEVEL_ALL);
      LogComponentEnable ("Icmpv6L4Protocol",
         LOG_LEVEL_ALL);
      LogComponentEnable ("NdiscCache", LOG_LEVEL_ALL
         );
      LogComponentEnable ("Ping6Application",
         LOG_LEVEL_ALL);
    }

//Enables either Split Horizon, No Split horizon or
   poison Reverse depending on what is selected
  if (showPings)
    {
      LogComponentEnable ("Ping6Application",
         LOG_LEVEL_INFO);
    }

  if (SplitHorizon == "NoSplitHorizon")
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::NO_SPLIT_HORIZON));
    }
  else if (SplitHorizon == "SplitHorizon")
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::SPLIT_HORIZON));
    }
  else
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::POISON_REVERSE));
    }

//To create the Nodes
  NS_LOG_INFO ("Create nodes.");
  Ptr<Node> src = CreateObject<Node> ();
//Associates the node with the name provided in the
   first argument
  Names::Add ("SrcNode", src);
  Ptr<Node> dst = CreateObject<Node> ();
  Names::Add ("DstNode", dst);
  Ptr<Node> a = CreateObject<Node> ();
  Names::Add ("RouterA", a);
  Ptr<Node> b = CreateObject<Node> ();
  Names::Add ("RouterB", b);
  Ptr<Node> c = CreateObject<Node> ();
  Names::Add ("RouterC", c);
  Ptr<Node> d = CreateObject<Node> ();
  Names::Add ("RouterD", d);
  Ptr<Node> e = CreateObject<Node> ();
  Names::Add ("RouterE", e);

//Defines a NodeContainer object that holds Devices
  NodeContainer net1 (src, a);
  NodeContainer net2 (a, b);
  NodeContainer net3 (b, c);
  NodeContainer net4 (b, d);
  NodeContainer net5 (c, d);
  NodeContainer net6 (c, e);
  NodeContainer net7 (d, e);
  NodeContainer net8 (e, dst);
  NodeContainer routers (a, b, c, d, e);
  NodeContainer nodes (src, dst);

//Creating CSMA channels
  NS_LOG_INFO ("Create channels.");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue
      (5000000));
  csma.SetChannelAttribute ("Delay", TimeValue (
     MilliSeconds (2)));

//Install CSMA Between the nodes
  NetDeviceContainer ndc1 = csma.Install (net1);
  NetDeviceContainer ndc2 = csma.Install (net2);
  NetDeviceContainer ndc3 = csma.Install (net3);
  NetDeviceContainer ndc4 = csma.Install (net4);
  NetDeviceContainer ndc5 = csma.Install (net5);
  NetDeviceContainer ndc6 = csma.Install (net6);
  NetDeviceContainer ndc7 = csma.Install (net7);
  NetDeviceContainer ndc8 = csma.Install (net8);
  NS_LOG_INFO ("Create IPv6 and routing");

//Defines an RipNgHelper object that helps configure
   RipNg
  RipNgHelper ripNgRouting;

//To exclude RIPng at the specified interface
//Interfaces start from 0 and are added sequentially
//However Interface 0 is for loopback address

  ripNgRouting.ExcludeInterface (a, 1);
  ripNgRouting.ExcludeInterface (e, 3);

//Sets the interface metric for the specified links
  ripNgRouting.SetInterfaceMetric (b, 2, 5);
  ripNgRouting.SetInterfaceMetric (c, 1, 5);
  ripNgRouting.SetInterfaceMetric (c, 3, 3);
  ripNgRouting.SetInterfaceMetric (e, 1, 3);

  Ipv6ListRoutingHelper listRH;
  listRH.Add (ripNgRouting, 0);

  InternetStackHelper internetv6;
  internetv6.SetIpv4StackInstall (false);
  internetv6.SetRoutingHelper (listRH);
  internetv6.Install (routers);

  InternetStackHelper internetv6Nodes;
  internetv6Nodes.SetIpv4StackInstall (false);
  internetv6Nodes.Install (nodes);
```

```
//Assigning IPv6 addresses
//Source and Destination have Global Addresses
//Rest of the routers need only link-local addresses
   for routing within the network
  NS_LOG_INFO ("Assign IPv6 Addresses.");
  Ipv6AddressHelper ipv6;

  ipv6.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix
     (64));
  Ipv6InterfaceContainer iic1 = ipv6.Assign (ndc1);
  iic1.SetForwarding (1, true);
  iic1.SetDefaultRouteInAllNodes (1);

  ipv6.SetBase (Ipv6Address ("2001:0:1::"),
     Ipv6Prefix (64));
  Ipv6InterfaceContainer iic2 = ipv6.Assign (ndc2);
  iic2.SetForwarding (0, true);
  iic2.SetForwarding (1, true);

  ipv6.SetBase (Ipv6Address ("2001:0:2::"),
     Ipv6Prefix (64));
  Ipv6InterfaceContainer iic3 = ipv6.Assign (ndc3);
  iic3.SetForwarding (0, true);
  iic3.SetForwarding (1, true);

  ipv6.SetBase (Ipv6Address ("2001:0:3::"),
     Ipv6Prefix (64));
  Ipv6InterfaceContainer iic4 = ipv6.Assign (ndc4);
  iic4.SetForwarding (0, true);
  iic4.SetForwarding (1, true);
```

```
ipv6.SetBase (Ipv6Address ("2001:0:4::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic5 = ipv6.Assign (ndc5);
iic5.SetForwarding (0, true);
iic5.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:5::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic6 = ipv6.Assign (ndc6);
iic6.SetForwarding (0, true);
iic6.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:6::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic7 = ipv6.Assign (ndc7);
iic7.SetForwarding (0, true);
iic7.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:2::"), Ipv6Prefix
  (64));
Ipv6InterfaceContainer iic8 = ipv6.Assign (ndc8);
iic8.SetForwarding (0, true);
iic8.SetDefaultRouteInAllNodes (0);

if (printRoutingTables)
  {
    RipNgHelper routingHelper;

    Ptr<OutputStreamWrapper> routingStream = Create
      <OutputStreamWrapper> (&std::cout);
```

```
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), a, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), b, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), c, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), d, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), e, routingStream);

routingHelper.PrintRoutingTableAt (Seconds
   (60.0), a, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (60.0), b, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (60.0), c, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (60.0), d, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (60.0), e, routingStream);

routingHelper.PrintRoutingTableAt (Seconds
   (90.0), a, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (90.0), b, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (90.0), c, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (90.0), d, routingStream);
```

```
      routingHelper.PrintRoutingTableAt (Seconds
         (90.0), e, routingStream);
    }

  NS_LOG_INFO ("Create Applications.");
  uint32_t packetSize = 1024;
  uint32_t maxPacketCount = 150;
  Time interPacketInterval = Seconds (1.0);
  Ping6Helper ping6;

//Set the source and destination address
  ping6.SetLocal (iic1.GetAddress (0, 1));
  ping6.SetRemote (iic8.GetAddress (1, 1));
  ping6.SetAttribute ("MaxPackets", UintegerValue (
     maxPacketCount));
  ping6.SetAttribute ("Interval", TimeValue (
     interPacketInterval));
  ping6.SetAttribute ("PacketSize", UintegerValue (
     packetSize));
  ApplicationContainer apps = ping6.Install (src);
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (160.0));

  AsciiTraceHelper ascii;
//To create the trace file
  csma.EnableAsciiAll (ascii.CreateFileStream ("ripng
     -3.tr"));
//Enables generation of .pcap files that can be
   examined using wireshark
  csma.EnablePcapAll ("ripng-3", true);
```

```
//Call the function defined above to tear down the
   link between two nodes
  Simulator::Schedule (Seconds (30), &TearDownLink, b
     , d, 3, 1);
  Simulator::Schedule (Seconds (75), &TearDownLink, c
     , d, 2, 2);

/* Now, do the actual simulation. */
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds (170));
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

3.4 RIPng Scenario-4

```
// Network topology
//
//
//     SRC
//      |  <=== Source Network
//      A       B
//       \     /|
//        \   / |
//         \/   |
//         /\   |
//        /  \  |
//       /    \|
//      D-------C
//      |  <=== Destination Network
//     DST
//
//
//
// Cost for the link from C to D is 10
// All other links are assumed to have cost 1
// A, B, C and D are RIPng routers.
// A and D are configured with static addresses.
// SRC and DST will exchange packets.
//
// After about 3 seconds, the topology is built, and
   Echo Reply will be received.
// After 30 seconds, the link between B and C will
   break, causing a route failure.
```

```
// After 12 seconds from the failure, the routers
   will recover from the failure.
// Split Horizoning should affect the recovery time,
   but it is not. See the manual
// for an explanation of this effect.
//
// If "showPings" is enabled, the user will see:
// 1) if the ping has been acknowledged
// 2) if a Destination Unreachable has been received
   by the sender
// 3) nothing, when the Echo Request has been
   received by the destination but
//    the Echo Reply is unable to reach the sender.
// Examining the .pcap files with Wireshark can
   confirm this effect.

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv6-static-routing-helper.h"
#include "ns3/ipv6-routing-table-entry.h"

using namespace ns3;

//Define a logging component for this file which can
   later be enabled or disabled with the NS_LOG
   environment variable
NS_LOG_COMPONENT_DEFINE ("RipNg-4");
```

```
//To tear down the link between any two interfaces
void TearDownLink (Ptr<Node> nodeA, Ptr<Node> nodeB,
   uint32_t interfaceA, uint32_t interfaceB)
{
  nodeA->GetObject<Ipv6> ()->SetDown (interfaceA);
  nodeB->GetObject<Ipv6> ()->SetDown (interfaceB);
}

int main (int argc, char **argv)
{
//Declare a verbose flag that can be set to true
   using the command line to enable log components
  bool verbose = false;
  bool printRoutingTables = false;
  bool showPings = false;
//Split horizon with Poison Reverse is used
  std::string SplitHorizon ("PoisonReverse");

//Creates an instance of CommandLine class to enable
   passsing command line arguments
  CommandLine cmd;
  cmd.AddValue ("verbose", "turn on log components",
     verbose);
  cmd.AddValue ("printRoutingTables", "Print routing
     tables at 30, 60 and 90 seconds",
     printRoutingTables);
  cmd.AddValue ("showPings", "Show Ping6 reception",
     showPings);
  cmd.AddValue ("splitHorizonStrategy", "Split
     Horizon strategy to use (NoSplitHorizon,
     SplitHorizon, PoisonReverse)", SplitHorizon);
```

```
//To parse the command line arguments
  cmd.Parse (argc, argv);

//Enables the Logging Components of all the levels
   above the level specified if verbose is set to true
   if (verbose)
     {
       LogComponentEnable ("RipNgSimpleRouting",
          LOG_LEVEL_INFO);
       LogComponentEnable ("RipNg", LOG_LEVEL_ALL);
       LogComponentEnable ("Icmpv6L4Protocol",
          LOG_LEVEL_INFO);
       LogComponentEnable ("Ipv6Interface",
          LOG_LEVEL_ALL);
       LogComponentEnable ("Icmpv6L4Protocol",
          LOG_LEVEL_ALL);
       LogComponentEnable ("NdiscCache", LOG_LEVEL_ALL
          );
       LogComponentEnable ("Ping6Application",
          LOG_LEVEL_ALL);
     }

   if (showPings)
     {
       LogComponentEnable ("Ping6Application",
          LOG_LEVEL_INFO);
     }

//Enables either Split Horizon, No Split horizon or
   poison Reverse depending on what is selected
   if (SplitHorizon == "NoSplitHorizon")
```

```
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::NO_SPLIT_HORIZON));
    }
  else if (SplitHorizon == "SplitHorizon")
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::SPLIT_HORIZON));
    }
  else
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::POISON_REVERSE));
    }

//To create the Nodes
  NS_LOG_INFO ("Create nodes.");
  Ptr<Node> src = CreateObject<Node> ();
//Associates the node with the name provided in the
   first argument
  Names::Add ("SrcNode", src);
  Ptr<Node> dst = CreateObject<Node> ();
  Names::Add ("DstNode", dst);
  Ptr<Node> a = CreateObject<Node> ();
  Names::Add ("RouterA", a);
  Ptr<Node> b = CreateObject<Node> ();
  Names::Add ("RouterB", b);
  Ptr<Node> c = CreateObject<Node> ();
  Names::Add ("RouterC", c);
  Ptr<Node> d = CreateObject<Node> ();
  Names::Add ("RouterD", d);
```

```
//Defines a NodeContainer object that holds Devices
  NodeContainer net1 (src, a);
  NodeContainer net2 (a, c);
  NodeContainer net3 (b, c);
  NodeContainer net4 (b, d);
  NodeContainer net5 (c, d);
  NodeContainer net6 (d, dst);
  NodeContainer routers (a, b, c, d);
  NodeContainer nodes (src, dst);

//Creating CSMA channels
  NS_LOG_INFO ("Create channels.");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue
     (5000000));
  csma.SetChannelAttribute ("Delay", TimeValue (
    MilliSeconds (2)));

//Install CSMA Between the nodes
  NetDeviceContainer ndc1 = csma.Install (net1);
  NetDeviceContainer ndc2 = csma.Install (net2);
  NetDeviceContainer ndc3 = csma.Install (net3);
  NetDeviceContainer ndc4 = csma.Install (net4);
  NetDeviceContainer ndc5 = csma.Install (net5);
  NetDeviceContainer ndc6 = csma.Install (net6);

  NS_LOG_INFO ("Create IPv6 and routing");

//Defines an RipNgHelper object that helps configure
  RipNg
  RipNgHelper ripNgRouting;

//To exclude RIPng at the specified interface
//Interfaces start from 0 and are added sequentially
//However Interface 0 is for loopback address
  ripNgRouting.ExcludeInterface (a, 1);
  ripNgRouting.ExcludeInterface (d, 3);

//Sets the interface metric for the specified links
  ripNgRouting.SetInterfaceMetric (c, 3, 10);
  ripNgRouting.SetInterfaceMetric (d, 2, 10);

  Ipv6ListRoutingHelper listRH;
  listRH.Add (ripNgRouting, 0);

  InternetStackHelper internetv6;
  internetv6.SetIpv4StackInstall (false);
  internetv6.SetRoutingHelper (listRH);
  internetv6.Install (routers);

  InternetStackHelper internetv6Nodes;
  internetv6Nodes.SetIpv4StackInstall (false);
  internetv6Nodes.Install (nodes);

//Assigning IPv6 addresses
//Source and Destination have Global Addresses
//Rest of the routers need only link-local addresses
   for routing within the network
  NS_LOG_INFO ("Assign IPv6 Addresses.");
  Ipv6AddressHelper ipv6;
```

```
ipv6.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix
  (64));
Ipv6InterfaceContainer iic1 = ipv6.Assign (ndc1);
iic1.SetForwarding (1, true);
iic1.SetDefaultRouteInAllNodes (1);

ipv6.SetBase (Ipv6Address ("2001:0:1::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic2 = ipv6.Assign (ndc2);
iic2.SetForwarding (0, true);
iic2.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:2::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic3 = ipv6.Assign (ndc3);
iic3.SetForwarding (0, true);
iic3.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:3::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic4 = ipv6.Assign (ndc4);
iic4.SetForwarding (0, true);
iic4.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:4::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic5 = ipv6.Assign (ndc5);
iic5.SetForwarding (0, true);
iic5.SetForwarding (1, true);
```

```
ipv6.SetBase (Ipv6Address ("2001:2::"), Ipv6Prefix
  (64));
Ipv6InterfaceContainer iic6 = ipv6.Assign (ndc6);
iic6.SetForwarding (0, true);
iic6.SetDefaultRouteInAllNodes (0);

if (printRoutingTables)
  {
    RipNgHelper routingHelper;

    Ptr<OutputStreamWrapper> routingStream = Create
      <OutputStreamWrapper> (&std::cout);

    routingHelper.PrintRoutingTableAt (Seconds
      (30.0), a, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (30.0), b, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (30.0), c, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (30.0), d, routingStream);

    routingHelper.PrintRoutingTableAt (Seconds
      (60.0), a, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (60.0), b, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (60.0), c, routingStream);
    routingHelper.PrintRoutingTableAt (Seconds
      (60.0), d, routingStream);
      routingHelper.PrintRoutingTableAt (Seconds
         (90.0), a, routingStream);
      routingHelper.PrintRoutingTableAt (Seconds
         (90.0), b, routingStream);
      routingHelper.PrintRoutingTableAt (Seconds
         (90.0), c, routingStream);
      routingHelper.PrintRoutingTableAt (Seconds
         (90.0), d, routingStream);
    }

  NS_LOG_INFO ("Create Applications.");
  uint32_t packetSize = 1024;
  uint32_t maxPacketCount = 100;
  Time interPacketInterval = Seconds (1.0);
  Ping6Helper ping6;

//Set the source and destination address
  ping6.SetLocal (iic1.GetAddress (0, 1));
  ping6.SetRemote (iic6.GetAddress (1, 1));
  ping6.SetAttribute ("MaxPackets", UintegerValue (
    maxPacketCount));
  ping6.SetAttribute ("Interval", TimeValue (
    interPacketInterval));
  ping6.SetAttribute ("PacketSize", UintegerValue (
    packetSize));
  ApplicationContainer apps = ping6.Install (src);
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (110.0));

  AsciiTraceHelper ascii;
//To create the trace file
```

```
  csma.EnableAsciiAll (ascii.CreateFileStream ("ripng
     -4.tr"));
//Enables generation of .pcap files that can be
   examined using wireshark
  csma.EnablePcapAll ("ripng-4", true);

//Call the function defined above to tear down the
   link between two nodes
  Simulator::Schedule (Seconds (30), &TearDownLink, b
     , c, 1, 2);

/* Now, do the actual simulation. */
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds (120));
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

3.5 RIPng Scenario-5

```
// Network topology
//
//
//       SRC
//        |  <=== Source Network
//        A-----------B
/*        |           |\
//        |           | \
//        |           |  \
//        C-----------D---E
//                        |  <=== Destination Network
//                       DST
*/
//
// Cost of the links are as follows:
// B to D : 2, C to D : 3
// All other links are assumed to have cost 1
// A, B, C, D and E are RIPng routers.
// A and E are configured with static addresses.
// SRC and DST will exchange packets.
//
// After about 3 seconds, the topology is built, and
   Echo Reply will be received.
// After 30 seconds, the link between B and E will
   break, causing a route failure.
// After 5 seconds from the failure, the routers will
    recovery from the failure.
// After 40 seconds from recovery, the link between B
    and D will break, causing a route failure.
```

```
// After 43 seconds from the failure, the routers
   will recover again from the failure.
// Split Horizoning should affect the recovery time,
   but it is not. See the manual
// for an explanation of this effect.
//
// If "showPings" is enabled, the user will see:
// 1) if the ping has been acknowledged
// 2) if a Destination Unreachable has been received
   by the sender
// 3) nothing, when the Echo Request has been
   received by the destination but
//    the Echo Reply is unable to reach the sender.
// Examining the .pcap files with Wireshark can
   confirm this effect.

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv6-static-routing-helper.h"
#include "ns3/ipv6-routing-table-entry.h"

using namespace ns3;

//Define a logging component for this file which can
   later be enabled or disabled with the NS_LOG
   environment variable
NS_LOG_COMPONENT_DEFINE ("RipNg-5");
```

```
//To tear down the link between any two interfaces
void TearDownLink (Ptr<Node> nodeA, Ptr<Node> nodeB,
   uint32_t interfaceA, uint32_t interfaceB)
{
  nodeA->GetObject<Ipv6> ()->SetDown (interfaceA);
  nodeB->GetObject<Ipv6> ()->SetDown (interfaceB);
}

int main (int argc, char **argv)
{
//Declare a verbose flag that can be set to true
   using the command line to enable log components
  bool verbose = false;
  bool printRoutingTables = false;
  bool showPings = false;
//Split horizon with Poison Reverse is used
  std::string SplitHorizon ("PoisonReverse");

//Creates an instance of CommandLine class to enable
   passsing command line arguments
  CommandLine cmd;
  cmd.AddValue ("verbose", "turn on log components",
     verbose);
  cmd.AddValue ("printRoutingTables", "Print routing
     tables at 30, 60 and 90 seconds",
     printRoutingTables);
  cmd.AddValue ("showPings", "Show Ping6 reception",
     showPings);
  cmd.AddValue ("splitHorizonStrategy", "Split
     Horizon strategy to use (NoSplitHorizon,
     SplitHorizon, PoisonReverse)", SplitHorizon);
```

```
//To parse the command line arguments
  cmd.Parse (argc, argv);

//Enables the Logging Components of all the levels
   above the level specified if verbose is set to true
  if (verbose)
    {
      LogComponentEnable ("RipNgSimpleRouting",
         LOG_LEVEL_INFO);
      LogComponentEnable ("RipNg", LOG_LEVEL_ALL);
      LogComponentEnable ("Icmpv6L4Protocol",
         LOG_LEVEL_INFO);
      LogComponentEnable ("Ipv6Interface",
         LOG_LEVEL_ALL);
      LogComponentEnable ("Icmpv6L4Protocol",
         LOG_LEVEL_ALL);
      LogComponentEnable ("NdiscCache", LOG_LEVEL_ALL
         );
      LogComponentEnable ("Ping6Application",
         LOG_LEVEL_ALL);
    }

  if (showPings)
    {
      LogComponentEnable ("Ping6Application",
         LOG_LEVEL_INFO);
    }

//Enables either Split Horizon, No Split horizon or
   poison Reverse depending on what is selected
  if (SplitHorizon == "NoSplitHorizon")
```

```
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::NO_SPLIT_HORIZON));
    }
  else if (SplitHorizon == "SplitHorizon")
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::SPLIT_HORIZON));
    }
  else
    {
      Config::SetDefault ("ns3::RipNg::SplitHorizon",
         EnumValue (RipNg::POISON_REVERSE));
    }

//To create the Nodes
  NS_LOG_INFO ("Create nodes.");
  Ptr<Node> src = CreateObject<Node> ();
//Associates the node with the name provided in the
   first argument
  Names::Add ("SrcNode", src);
  Ptr<Node> dst = CreateObject<Node> ();
  Names::Add ("DstNode", dst);
  Ptr<Node> a = CreateObject<Node> ();
  Names::Add ("RouterA", a);
  Ptr<Node> b = CreateObject<Node> ();
  Names::Add ("RouterB", b);
  Ptr<Node> c = CreateObject<Node> ();
  Names::Add ("RouterC", c);
  Ptr<Node> d = CreateObject<Node> ();
  Names::Add ("RouterD", d);
  Ptr<Node> e = CreateObject<Node> ();
  Names::Add ("RouterE", e);

//Defines a NodeContainer object that holds Devices
  NodeContainer net1 (src, a);
  NodeContainer net2 (a, b);
  NodeContainer net3 (a, c);
  NodeContainer net4 (b, d);
  NodeContainer net5 (b, e);
  NodeContainer net6 (c, d);
  NodeContainer net7 (d, e);
  NodeContainer net8 (e, dst);
  NodeContainer routers (a, b, c, d, e);
  NodeContainer nodes (src, dst);

//Creating CSMA channels
  NS_LOG_INFO ("Create channels.");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue
      (5000000));
  csma.SetChannelAttribute ("Delay", TimeValue (
     MilliSeconds (2)));

//Install CSMA Between the nodes
  NetDeviceContainer ndc1 = csma.Install (net1);
  NetDeviceContainer ndc2 = csma.Install (net2);
  NetDeviceContainer ndc3 = csma.Install (net3);
  NetDeviceContainer ndc4 = csma.Install (net4);
  NetDeviceContainer ndc5 = csma.Install (net5);
  NetDeviceContainer ndc6 = csma.Install (net6);
  NetDeviceContainer ndc7 = csma.Install (net7);
  NetDeviceContainer ndc8 = csma.Install (net8);
  NS_LOG_INFO ("Create IPv6 and routing");
//Defines an RipNgHelper object that helps configure
   RipNg
  RipNgHelper ripNgRouting;

//To exclude RIPng at the specified interface
//Interfaces start from 0 and are added sequentially
//However Interface 0 is for loopback address

  ripNgRouting.ExcludeInterface (a, 1);
  ripNgRouting.ExcludeInterface (e, 3);

//Sets the interface metric for the specified links
  ripNgRouting.SetInterfaceMetric (b, 2, 2);
  ripNgRouting.SetInterfaceMetric (d, 1, 2);
  ripNgRouting.SetInterfaceMetric (c, 2, 3);
  ripNgRouting.SetInterfaceMetric (d, 2, 3);

  Ipv6ListRoutingHelper listRH;
  listRH.Add (ripNgRouting, 0);

  InternetStackHelper internetv6;
  internetv6.SetIpv4StackInstall (false);
  internetv6.SetRoutingHelper (listRH);
  internetv6.Install (routers);

  InternetStackHelper internetv6Nodes;
  internetv6Nodes.SetIpv4StackInstall (false);
  internetv6Nodes.Install (nodes);
```

```
//Assigning IPv6 addresses
//Source and Destination have Global Addresses
//Rest of the routers need only link-local addresses
   for routing within the network

  NS_LOG_INFO ("Assign IPv6 Addresses.");
  Ipv6AddressHelper ipv6;

  ipv6.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix
    (64));
  Ipv6InterfaceContainer iic1 = ipv6.Assign (ndc1);
  iic1.SetForwarding (1, true);
  iic1.SetDefaultRouteInAllNodes (1);

  ipv6.SetBase (Ipv6Address ("2001:0:1::"),
    Ipv6Prefix (64));
  Ipv6InterfaceContainer iic2 = ipv6.Assign (ndc2);
  iic2.SetForwarding (0, true);
  iic2.SetForwarding (1, true);

  ipv6.SetBase (Ipv6Address ("2001:0:2::"),
    Ipv6Prefix (64));
  Ipv6InterfaceContainer iic3 = ipv6.Assign (ndc3);
  iic3.SetForwarding (0, true);
  iic3.SetForwarding (1, true);

  ipv6.SetBase (Ipv6Address ("2001:0:3::"),
    Ipv6Prefix (64));
  Ipv6InterfaceContainer iic4 = ipv6.Assign (ndc4);
  iic4.SetForwarding (0, true);
  iic4.SetForwarding (1, true);
```

```
ipv6.SetBase (Ipv6Address ("2001:0:4::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic5 = ipv6.Assign (ndc5);
iic5.SetForwarding (0, true);
iic5.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:5::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic6 = ipv6.Assign (ndc6);
iic6.SetForwarding (0, true);
iic6.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:0:6::"),
  Ipv6Prefix (64));
Ipv6InterfaceContainer iic7 = ipv6.Assign (ndc7);
iic7.SetForwarding (0, true);
iic7.SetForwarding (1, true);

ipv6.SetBase (Ipv6Address ("2001:2::"), Ipv6Prefix
  (64));
Ipv6InterfaceContainer iic8 = ipv6.Assign (ndc8);
iic8.SetForwarding (0, true);
iic8.SetDefaultRouteInAllNodes (0);

if (printRoutingTables)
  {
    RipNgHelper routingHelper;

    Ptr<OutputStreamWrapper> routingStream = Create
      <OutputStreamWrapper> (&std::cout);
```

```
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), a, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), b, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), c, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), d, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (30.0), e, routingStream);

routingHelper.PrintRoutingTableAt (Seconds
   (60.0), a, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (60.0), b, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (60.0), c, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (60.0), d, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds
        (60.0), e, routingStream);

routingHelper.PrintRoutingTableAt (Seconds
   (90.0), a, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (90.0), b, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (90.0), c, routingStream);
routingHelper.PrintRoutingTableAt (Seconds
   (90.0), d, routingStream);
```

```
          routingHelper.PrintRoutingTableAt (Seconds
            (90.0), e, routingStream);
      }

  NS_LOG_INFO ("Create Applications.");
  uint32_t packetSize = 1024;
  uint32_t maxPacketCount = 150;
  Time interPacketInterval = Seconds (1.0);
  Ping6Helper ping6;

//Set the source and destination address
  ping6.SetLocal (iic1.GetAddress (0, 1));
  ping6.SetRemote (iic8.GetAddress (1, 1));
  ping6.SetAttribute ("MaxPackets", UintegerValue (
    maxPacketCount));
  ping6.SetAttribute ("Interval", TimeValue (
    interPacketInterval));
  ping6.SetAttribute ("PacketSize", UintegerValue (
    packetSize));
  ApplicationContainer apps = ping6.Install (src);
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (160.0));

  AsciiTraceHelper ascii;

//To create the trace file
  csma.EnableAsciiAll (ascii.CreateFileStream ("ripng
    -5.tr"));
//Enables generation of .pcap files that can be
  examined using wireshark
  csma.EnablePcapAll ("ripng-5", true);
```

```
//Call the function defined above to tear down the
   link between two nodes
  Simulator::Schedule (Seconds (30), &TearDownLink, b
     , e, 3, 1);
  Simulator::Schedule (Seconds (75), &TearDownLink, b
     , d, 2, 1);

/* Now, do the actual simulation. */
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds (170));
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

4 INFERENCE:

From the above Simulations, We were able to understand how RIPng works in an IPv6 Network. Also, we were able to see how Split Horizoning works in case of Route failure. Some observations made about RIPng are as follows: It is ideal for networks diameter smaller than 15. Split Horizon And Poison Reverse Are Used To Prevent Routing Loops that may occur when a link goes down. The path is calculated using Bellman-Ford algorithm and routing is carried out. RIPng is almost similar to RIPv2 with minimal changes done just to support IPv6 networking