**MESH 1**

/\* -\*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -\*- \*/

/\*

\* Copyright (c) 2008,2009 IITP RAS

\*

\* This program is free software; you can redistribute it and/or modify

\* it under the terms of the GNU General Public License version 2 as

\* published by the Free Software Foundation;

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program; if not, write to the Free Software

\* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*

\* Author: Kirill Andreev <andreev@iitp.ru>

\*

\*

\* By default this script creates m\_xSize \* m\_ySize square grid topology with

\* IEEE802.11s stack installed at each node with peering management

\* and HWMP protocol.

\* The side of the square cell is defined by m\_step parameter.

\* When topology is created, UDP ping is installed to opposite corners

\* by diagonals. packet size of the UDP ping and interval between two

\* successive packets is configurable.

\*

\* m\_xSize \* step

\* |<--------->|

\* step

\* |<--->|

\* \* --- \* --- \* <---Ping sink \_

\* | \ | / | ^

\* | \ | / | |

\* \* --- \* --- \* m\_ySize \* step |

\* | / | \ | |

\* | / | \ | |

\* \* --- \* --- \* \_

\* ^ Ping source

\*

\* See also MeshTest::Configure to read more about configurable

\* parameters.

\*/

#include "ns3/core-module.h"

#include "ns3/internet-module.h"

#include "ns3/network-module.h"

#include "ns3/applications-module.h"

#include "ns3/wifi-module.h"

#include "ns3/mesh-module.h"

#include "ns3/mobility-module.h"

#include "ns3/mesh-helper.h"

#include <iostream>

#include <sstream>

#include <fstream>

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("TestMeshScript");

class MeshTest

{

public:

MeshTest ();

void Configure (int argc, char \*\* argv);

int Run ();

private:

int m\_xSize;

int m\_ySize;

double m\_step;

double m\_randomStart;

double m\_totalTime;

double m\_packetInterval;

uint16\_t m\_packetSize;

uint32\_t m\_nIfaces;

bool m\_chan;

bool m\_pcap;

std::string m\_stack;

std::string m\_root;

NodeContainer nodes;

NetDeviceContainer meshDevices;

// Addresses of interfaces:

Ipv4InterfaceContainer interfaces;

// MeshHelper. Report is not static methods

MeshHelper mesh;

private:

void CreateNodes ();

void InstallInternetStack ();

void InstallApplication ();

void Report ();

};

MeshTest::MeshTest () :

m\_xSize (3),

m\_ySize (3),

m\_step (100.0),

m\_randomStart (0.1),

m\_totalTime (100.0),

m\_packetInterval (0.1),

m\_packetSize (1024),

m\_nIfaces (1),

m\_chan (true),

m\_pcap (false),

m\_stack ("ns3::Dot11sStack"),

m\_root ("ff:ff:ff:ff:ff:ff")

{

}

void

MeshTest::Configure (int argc, char \*argv[])

{

CommandLine cmd;

cmd.AddValue ("x-size", "Number of nodes in a row grid. [6]", m\_xSize);

cmd.AddValue ("y-size", "Number of rows in a grid. [6]", m\_ySize);

cmd.AddValue ("step", "Size of edge in our grid, meters. [100 m]", m\_step);

/\*

\* As soon as starting node means that it sends a beacon,

\* simultaneous start is not good.

\*/

cmd.AddValue ("start", "Maximum random start delay, seconds. [0.1 s]", m\_randomStart);

cmd.AddValue ("time", "Simulation time, seconds [100 s]", m\_totalTime);

cmd.AddValue ("packet-interval", "Interval between packets in UDP ping, seconds [0.001 s]", m\_packetInterval);

cmd.AddValue ("packet-size", "Size of packets in UDP ping", m\_packetSize);

cmd.AddValue ("interfaces", "Number of radio interfaces used by each mesh point. [1]", m\_nIfaces);

cmd.AddValue ("channels", "Use different frequency channels for different interfaces. [0]", m\_chan);

cmd.AddValue ("pcap", "Enable PCAP traces on interfaces. [0]", m\_pcap);

cmd.AddValue ("stack", "Type of protocol stack. ns3::Dot11sStack by default", m\_stack);

cmd.AddValue ("root", "Mac address of root mesh point in HWMP", m\_root);

cmd.Parse (argc, argv);

NS\_LOG\_DEBUG ("Grid:" << m\_xSize << "\*" << m\_ySize);

NS\_LOG\_DEBUG ("Simulation time: " << m\_totalTime << " s");

}

void

MeshTest::CreateNodes ()

{

/\*

\* Create m\_ySize\*m\_xSize stations to form a grid topology

\*/

nodes.Create (m\_ySize\*m\_xSize);

// Configure YansWifiChannel

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();

YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();

wifiPhy.SetChannel (wifiChannel.Create ());

/\*

\* Create mesh helper and set stack installer to it

\* Stack installer creates all needed protocols and installs them to

\* mesh point device

\*/

mesh = MeshHelper::Default ();

if (!Mac48Address (m\_root.c\_str ()).IsBroadcast ())

{

mesh.SetStackInstaller (m\_stack, "Root", Mac48AddressValue (Mac48Address (m\_root.c\_str ())));

}

else

{

// If root is not set, we do not use "Root" attribute, because it

// is specified only for 11s

mesh.SetStackInstaller (m\_stack);

}

if (m\_chan)

{

mesh.SetSpreadInterfaceChannels (MeshHelper::SPREAD\_CHANNELS);

}

else

{

mesh.SetSpreadInterfaceChannels (MeshHelper::ZERO\_CHANNEL);

}

mesh.SetMacType ("RandomStart", TimeValue (Seconds (m\_randomStart)));

// Set number of interfaces - default is single-interface mesh point

mesh.SetNumberOfInterfaces (m\_nIfaces);

// Install protocols and return container with MeshPointDevices

meshDevices = mesh.Install (wifiPhy, nodes);

// Setup mobility - static grid topology

MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",

"MinX", DoubleValue (0.0),

"MinY", DoubleValue (0.0),

"DeltaX", DoubleValue (m\_step),

"DeltaY", DoubleValue (m\_step),

"GridWidth", UintegerValue (m\_xSize),

"LayoutType", StringValue ("RowFirst"));

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

mobility.Install (nodes);

if (m\_pcap)

{

wifiPhy.EnablePcapAll (std::string ("mp-"));

}

}

void

MeshTest::InstallInternetStack ()

{

InternetStackHelper internetStack;

internetStack.Install (nodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");

interfaces = address.Assign (meshDevices);

}

void

MeshTest::InstallApplication ()

{

// UdpEchoServerHelper echoServer (9);

// ApplicationContainer serverApps = echoServer.Install (nodes.Get (0));

// serverApps.Start (Seconds (0.0));

// serverApps.Stop (Seconds (m\_totalTime));

// UdpEchoClientHelper echoClient (interfaces.GetAddress (0), 9);

// echoClient.SetAttribute ("MaxPackets", UintegerValue ((uint32\_t)(m\_totalTime\*(1/m\_packetInterval))));

// echoClient.SetAttribute ("Interval", TimeValue (Seconds (m\_packetInterval)));

// echoClient.SetAttribute ("PacketSize", UintegerValue (m\_packetSize));

// ApplicationContainer clientApps = echoClient.Install (nodes.Get (m\_xSize \* m\_ySize - 1));

// clientApps.Start (Seconds (0.0));

// clientApps.Stop (Seconds (m\_totalTime));

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (Ipv4Address ("10.1.1.2"), 9);

echoClient.SetAttribute ("MaxPackets", UintegerValue (1));

echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));

clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));

}

int

MeshTest::Run ()

{

CreateNodes ();

InstallInternetStack ();

InstallApplication ();

Simulator::Schedule (Seconds (m\_totalTime), &MeshTest::Report, this);

Simulator::Stop (Seconds (m\_totalTime));

Simulator::Run ();

Simulator::Destroy ();

return 0;

}

void

MeshTest::Report ()

{

unsigned n (0);

for (NetDeviceContainer::Iterator i = meshDevices.Begin (); i != meshDevices.End (); ++i, ++n)

{

std::ostringstream os;

os << "mp-report-" << n << ".xml";

std::cerr << "Printing mesh point device #" << n << " diagnostics to " << os.str () << "\n";

std::ofstream of;

of.open (os.str ().c\_str ());

if (!of.is\_open ())

{

std::cerr << "Error: Can't open file " << os.str () << "\n";

return;

}

mesh.Report (\*i, of);

of.close ();

}

}

int

main (int argc, char \*argv[])

{

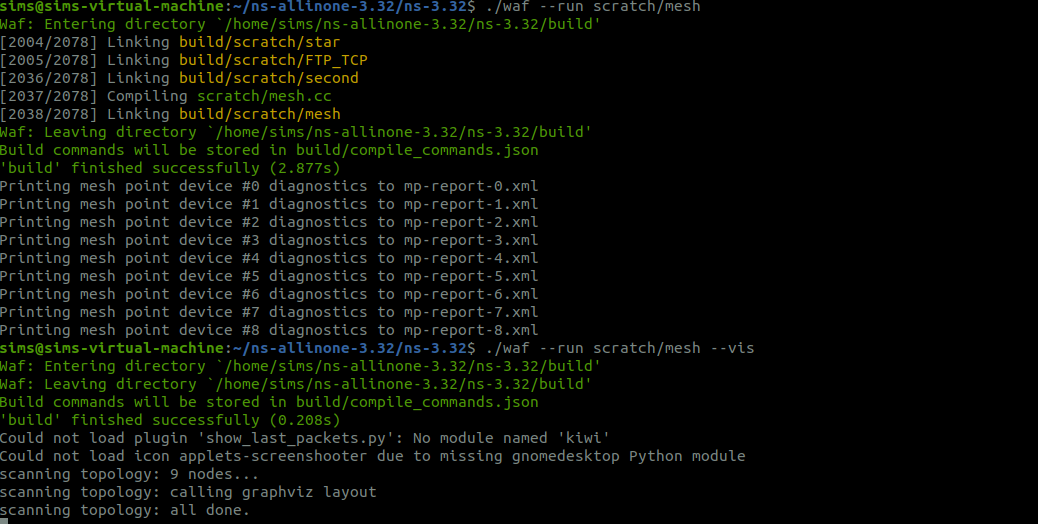
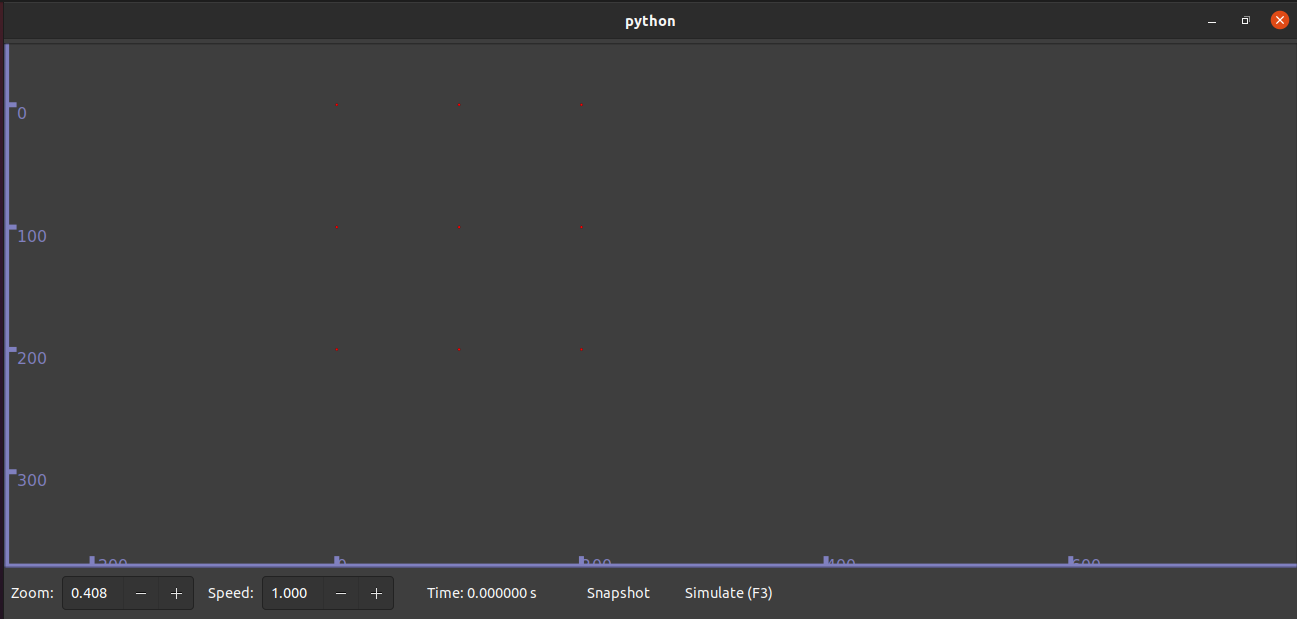
MeshTest t;

t.Configure (argc, argv);

return t.Run ();

}

OUTPUT



**MESH 2**

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/mobility-module.h"

#include "ns3/wifi-module.h"

#include "ns3/applications-module.h"

#include "ns3/mesh-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("MeshExample");

int main(int argc, char \*argv[])

{

// Enable logging

LogComponentEnable("MeshExample", LOG\_LEVEL\_INFO);

LogComponentEnable("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

LogComponentEnable("UdpEchoServerApplication", LOG\_LEVEL\_INFO);

LogComponentEnable("WifiMac", LOG\_LEVEL\_INFO);

// Set up command line arguments

CommandLine cmd;

cmd.Parse(argc, argv);

// Create 4 nodes for the mesh network

NodeContainer nodes;

nodes.Create(4);

// Set up Wi-Fi devices (mesh network)

WifiMacHelper wifiMac;

wifiMac.SetType("ns3::AdhocWifiMac");

WifiHelper wifi;

wifi.SetRemoteStationManager("ns3::AarfWifiManager");

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();

YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();

wifiPhy.SetChannel(wifiChannel.Create());

NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, nodes);

// Set up Mobility Model

MobilityHelper mobility;

// Use ConstantPositionMobilityModel to avoid freezing

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");

// Simple position allocator - placing nodes in a grid

Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator>();

positionAlloc->Add(Vector(10.0, 10.0, 0.0));

positionAlloc->Add(Vector(20.0, 10.0, 0.0));

positionAlloc->Add(Vector(10.0, 20.0, 0.0));

positionAlloc->Add(Vector(20.0, 20.0, 0.0));

mobility.SetPositionAllocator(positionAlloc);

mobility.Install(nodes);

// Install Internet stack

InternetStackHelper stack;

stack.Install(nodes);

// Set up IP addresses for nodes

Ipv4AddressHelper address;

address.SetBase("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign(devices);

// Create and install the UDP server application on node 1 (server)

UdpEchoServerHelper echoServer(9);

ApplicationContainer serverApps = echoServer.Install(nodes.Get(1)); // Server on node 1

serverApps.Start(Seconds(1.0));

serverApps.Stop(Seconds(10.0));

// Create and install the UDP client application on node 0 (client)

UdpEchoClientHelper echoClient(interfaces.GetAddress(1), 9);

echoClient.SetAttribute("MaxPackets", UintegerValue(10));

echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));

echoClient.SetAttribute("PacketSize", UintegerValue(1024));

ApplicationContainer clientApps = echoClient.Install(nodes.Get(0)); // Client on node 0

clientApps.Start(Seconds(2.0));

clientApps.Stop(Seconds(10.0));

// Run the simulation

Simulator::Run();

Simulator::Destroy();

return 0;

}

OUTPUT

