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

/\*

\* 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

\*/

#include "ns3/core-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/network-module.h"

#include "ns3/applications-module.h"

#include "ns3/mobility-module.h"

#include "ns3/csma-module.h"

#include "ns3/internet-module.h"

#include "ns3/yans-wifi-helper.h"

#include "ns3/ssid.h"

// Default Network Topology

//

// Wifi 10.1.3.0

// AP

// \* \* \* \*

// | | | | 10.1.1.0

// n5 n6 n7 n0 -------------- n1 n2 n3 n4

// point-to-point | | | |

// ================

// LAN 10.1.2.0

using namespace ns3;

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

int

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

{

bool verbose = true;

uint32\_t nCsma = 3;

uint32\_t nWifi = 3;

bool tracing = false;

CommandLine cmd (\_\_FILE\_\_);

cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);

cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);

cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

cmd.Parse (argc,argv);

// The underlying restriction of 18 is due to the grid position

// allocator's configuration; the grid layout will exceed the

// bounding box if more than 18 nodes are provided.

if (nWifi > 18)

{

std::cout << "nWifi should be 18 or less; otherwise grid layout exceeds the bounding box" << std::endl;

return 1;

}

if (verbose)

{

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

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

}

NodeContainer p2pNodes;

p2pNodes.Create (2);

PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;

csmaNodes.Add (p2pNodes.Get (1));

csmaNodes.Create (nCsma);

CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));

csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;

csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;

wifiStaNodes.Create (nWifi);

NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();

YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();

phy.SetChannel (channel.Create ());

WifiHelper wifi;

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

WifiMacHelper mac;

Ssid ssid = Ssid ("ns-3-ssid");

mac.SetType ("ns3::StaWifiMac",

"Ssid", SsidValue (ssid),

"ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;

staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac",

"Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;

apDevices = wifi.Install (phy, mac, wifiApNode);

MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",

"MinX", DoubleValue (0.0),

"MinY", DoubleValue (0.0),

"DeltaX", DoubleValue (5.0),

"DeltaY", DoubleValue (10.0),

"GridWidth", UintegerValue (3),

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

mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",

"Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));

mobility.Install (wifiStaNodes);

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

mobility.Install (wifiApNode);

InternetStackHelper stack;

stack.Install (csmaNodes);

stack.Install (wifiApNode);

stack.Install (wifiStaNodes);

Ipv4AddressHelper address;

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

Ipv4InterfaceContainer p2pInterfaces;

p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer csmaInterfaces;

csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");

address.Assign (staDevices);

address.Assign (apDevices);

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);

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

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

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

ApplicationContainer clientApps =

echoClient.Install (wifiStaNodes.Get (nWifi - 1));

clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

if (tracing == true)

{

pointToPoint.EnablePcapAll ("third");

phy.EnablePcap ("third", apDevices.Get (0));

csma.EnablePcap ("third", csmaDevices.Get (0), true);

}

Simulator::Run ();

Simulator::Destroy ();

return 0;

}

OUTPUT:

