```
/*
 * 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/mobility-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/bridge-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/olsr-helper.h"
#include <vector>
#include <stdint.h>
#include <sstream>
#include <fstream>

using namespace ns3;

int main (int argc, char *argv[])
{
     uint32_t nWifis = 2;     // Número de APs
     uint32_t nStas = 5; // Número de clientes
     uint32_t flowType = 0;   // Tipo do tráfego
     uint32_t simTime = 60.0; //Tempo de simulação

     // Flags para a linha de execução
     CommandLine cmd;
     cmd.AddValue ("nStas", "Numero de clientes", nStas);
     cmd.AddValue ("flowType", "Tipo do fluxo, 0 para CBR e 1 para rajadas", flowType);
     cmd.AddValue ("simTime", "Tempo de simulação", simTime);
     cmd.Parse (argc, argv);

     // Grupos dos nós da rede
     NodeContainer apNodes;
     NodeContainer staNodes;
     NodeContainer csmaSwitch;
     NodeContainer serverNodes;

     // Grupos das insterfaces da rede
     NetDeviceContainer apDevices;
     NetDeviceContainer switchDevices;
     NetDeviceContainer serverDevices;
     NetDeviceContainer staDevices;

     // Grupos das subnets da rede
     Ipv4InterfaceContainer apInterfaces;
     Ipv4InterfaceContainer serverInterfaces;
     Ipv4InterfaceContainer staInterfaces;

     // Pilha dos protocolos e buffers
     InternetStackHelper stack;

     // Construtor do CSMA
     CsmaHelper csma;

     // Ips bases das subnets
     Ipv4AddressHelper ipServer, ipSta;
     ipServer.SetBase ("192.168.0.0", "255.255.255.0");
     ipSta.SetBase ("192.168.1.0", "255.255.255.0");

```

```
75        // Criação dos nós da rede
76        apNodes.Create(nWifis);
77        staNodes.Create(nStas);
78        serverNodes.Create (1);
79        csmaSwitch.Create (1);
80
81        // Configuração dos parâmetros do CSMA
82        NetDeviceContainer link;
83        csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
84        csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
85
86        // Conecta os APs no switch
87        switchDevices = csma.Install (NodeContainer (csmaSwitch, apNodes));
88        // Conecta o servidor no switch
89        serverDevices = csma.Install(NodeContainer(serverNodes, csmaSwitch));
90
91        // Configuração dos parâmetros da WIFI
92        YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
93        wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
94        NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
95        YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
96        wifiPhy.SetChannel (wifiChannel.Create ());
97
98        // Construtor das posições e mobilidade dos nós
99        MobilityHelper mobility;
100
101       // Primeiro AP no ponto (0,0)
102       mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
103                       "MinX", DoubleValue (0.0),
104                       "MinY", DoubleValue (0.0),
105                       "DeltaX", DoubleValue (5.0),
106                       "DeltaY", DoubleValue (5.0),
107                       "GridWidth", UintegerValue (1),
108                       "LayoutType", StringValue ("RowFirst"));
109       // Primeiro AP com posição constante
110       mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
111       // Instala no primeiro AP
112       mobility.Install (apNodes.Get(0));
113
114       // Segundo AP no ponto (100,100)
115       mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
116                       "MinX", DoubleValue (100.0),
117                       "MinY", DoubleValue (100.0),
118                       "DeltaX", DoubleValue (5.0),
119                       "DeltaY", DoubleValue (5.0),
120                       "GridWidth", UintegerValue (1),
121                       "LayoutType", StringValue ("RowFirst"));
122       // Segudno AP com posição constante
123       mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
124       // Instala no segundo AP
125       mobility.Install (apNodes.Get(1));
126
127       // Posiciona o servidor e o switch em posições irrelevantes
128       mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
129                       "MinX", DoubleValue (50.0),
130                       "MinY", DoubleValue (50.0),
131                       "DeltaX", DoubleValue (50.0),
132                       "DeltaY", DoubleValue (50.0),
133                       "GridWidth", UintegerValue (1.0),
134                       "LayoutType", StringValue ("RowFirst"));
135       // Servidor e switch com posição constante
136       mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
137       // Instala no servidor e no switch
138       mobility.Install (NodeContainer(serverNodes, csmaSwitch));
139
140       // Aloca um quandrado de lado 100m e posiciona os clientes em posições aleatórias
141       mobility.SetPositionAllocator ("ns3::RandomRectanglePositionAllocator",
142                 "X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=100.0]"),
143                 "Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=100.0]"));
144       // Movimenta os clientes em direções aleatórias
145       mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
146                       "Mode", StringValue ("Time"),
147                       "Time", StringValue ("2s"),
148                       "Speed", StringValue ("ns3::ConstantRandomVariable[Constant=10.0]"));
```

```
149       // Instala no clientes
150       mobility.Install (staNodes);
151
152       // SSID da rede WIFI
153       std::ostringstream oss;
154       oss << "wifi-default";
155       Ssid ssid = Ssid (oss.str ());
156
157       WifiHelper wifi = WifiHelper::Default ();
158
159       // Configura a WIFI dos APs
160       wifiMac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
161       // Instala a WIFI nos APs
162       apDevices = wifi.Install (wifiPhy, wifiMac, apNodes);
163
164       // Cria uma Bridge entre o switch e os APs
165       Ptr<Node> switchNode = csmaSwitch.Get(0);
166       BridgeHelper bridge;
167       NetDeviceContainer bridgeDevices;
168       bridgeDevices.Add (bridge.Install(apNodes.Get (0), NetDeviceContainer (apDevices.Get (0),
      switchDevices.Get (1))));
169       bridgeDevices.Add (bridge.Install(apNodes.Get (1), NetDeviceContainer (apDevices.Get (1),
      switchDevices.Get (2))));
170
171       // Configura a WIFI nos clientes
172       wifiMac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid));
173       // Instala a WIFI nos clientes
174       staDevices = wifi.Install (wifiPhy, wifiMac, staNodes);
175
176       // Altera o algoritmo de roteamento para o OLSR
177       OlsrHelper olsr;
178       stack.SetRoutingHelper(olsr);
179
180       // Instala as pilhas e os buffers nos nós
181       stack.Install (serverNodes);
182       stack.Install (apNodes);
183       stack.Install (staNodes);
184       stack.Install (csmaSwitch);
185
186       // Atribui IP para o servidor
187       serverInterfaces = ipServer.Assign(NetDeviceContainer(serverDevices));
188       // Atribui IP para o switch, as bridges e os clientes
189       staInterfaces.Add(ipSta.Assign(switchDevices.Get(0)));
190       staInterfaces.Add(ipSta.Assign(bridgeDevices));
191       staInterfaces.Add(ipSta.Assign(staDevices));
192
193       // Configura a aplicação
194       Address dest;
195       std::string protocol;
196       dest = InetSocketAddress (serverInterfaces.GetAddress(0), 1025);
197       protocol = "ns3::UdpSocketFactory";
198
199       OnOffHelper onoff = OnOffHelper (protocol, dest);
200       if(flowType == 0) { // Tráfego CBR
201            // Pacotes de 512 bytes
202            onoff.SetAttribute ("PacketSize", UintegerValue (512));
203            // Sempre envia pacotes
204            onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable
      [Constant=0.0]"));
205       }
206       else {    // Tráfego em rajadas
207            // Pacotes de 1500 bytes
208            onoff.SetAttribute ("PacketSize", UintegerValue (1500));
209            // Envia rajadas entre intervalos de silêncio
210            onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable
      [Constant=1.0]"));
211            onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable
      [Constant=0.1]"));
212       }
213
214       // Instala a aplicação nos clientes
215       ApplicationContainer apps = onoff.Install (staNodes);
216
217       // Tempo de início da aplicação
```

```
218       apps.Start (Seconds (1.5));
219       apps.Stop (Seconds (simTime));
220
221       // Popula as tabelas de roteamento
222       Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
223
224       // FlowMonitor, para colher as informações
225       FlowMonitorHelper flowmon;
226       Ptr<FlowMonitor> monitor = flowmon.InstallAll();
227
228       // Tempo de término da simulação
229       Simulator::Stop (Seconds (simTime));
230       // Roda a simulação
231       Simulator::Run ();
232
233       // Itera nas informações obtidas da simulação para imprimir os dados requeridos
234       monitor->CheckForLostPackets ();
235       Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier
      ());
236       FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats ();
237       for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i !=
     stats.end (); ++i)
238       {
239            {
240                 Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
241                 std::cout << "Flow " << i->first << " (" << t.sourceAddress << " -> " <<
     t.destinationAddress << ")\n";
242                 std::cout << "  Tx Packets: " << i->second.txPackets << "\n";
243                 std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";
244                 std::cout << "  TxOffered:  " << i->second.txBytes * 8.0 / 9.0 / 1000 / 1000  <<
     " Mbps\n";
245                 std::cout << "  Rx Packets: " << i->second.rxPackets << "\n";
246                 std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
247                 std::cout << "  Throughput: " << i->second.rxBytes * 8.0 / 9.0 / 1000 / 1000  <<
     " Mbps\n";
248                 std::cout << "  Lost Packets: " << i->second.lostPackets << "\n";
249                 std::cout << "  Delay: " << i->second.delaySum / i->second.rxPackets / 1000 /
     1000 << "\n";
250            }
251       }
252
253       // Destrói a simulação
254       Simulator::Destroy ();
255  }
```