```
00001: /* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
00002: /*
00003:  * Copyright (c) 2011 The Boeing Company
00004:  *
00005:  * This program is free software; you can redistribute it and/or modify
00006:  * it under the terms of the GNU General Public License version 2 as
00007:  * published by the Free Software Foundation;
00008:  *
00009:  * This program is distributed in the hope that it will be useful,
00010:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012:  * GNU General Public License for more details.
00013:  *
00014:  * You should have received a copy of the GNU General Public License
00015:  * along with this program; if not, write to the Free Software
00016:  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
00017:  *
00018:  * Author:  Tom Henderson <thomas.r.henderson@boeing.com>
00019:  */
00020:
00021: /*
00022:  * Try to send data end-to-end through a LrWpanMac <-> LrWpanPhy <->
00023:  * SpectrumChannel <-> LrWpanPhy <-> LrWpanMac chain
00024:  *
00025:  * Trace Phy state changes, and Mac DataIndication and DataConfirm events
00026:  * to stdout
00027:  */
00028: #include <ns3/log.h>
00029: #include <ns3/core-module.h>
00030: #include <ns3/lr-wpan-module.h>
00031: #include <ns3/propagation-loss-model.h>
00032: #include <ns3/propagation-delay-model.h>
00033: #include <ns3/simulator.h>
00034: #include <ns3/single-model-spectrum-channel.h>
00035: #include <ns3/constant-position-mobility-model.h>
00036: #include <ns3/packet.h>
00037: #include <ns3/mobility-module.h>
00038: #include <cmath>
00039:
00040:
00041: #include <iostream>
00042:
00043: using namespace ns3;
00044:
00045: uint32_t RECV_STATIC=0;
00046: static void DataIndication (McpsDataIndicationParams params, Ptr<Packet> p)
00047: {
00048:   //NS_LOG_UNCOND ("Received packet of size " << p->GetSize ());
00049:   RECV_STATIC++;
00050: }
00051:
00052: static void DataConfirm (McpsDataConfirmParams params)
00053: {
00054:  /*
00055:  NS_LOG_UNCOND ("LrWpanMcpsDataConfirmStatus = " << params.m_status);
00056:  */
00057: }
00058:
00059: static void StateChangeNotification (std::string context, Time now, LrWpanPhyEnumeration
00059: oldState, LrWpanPhyEnumeration newState)
00060: {
00061: /*
00062:  NS_LOG_UNCOND (context << " state change at " << now.GetSeconds ()
00063:                 << " from " << LrWpanHelper::LrWpanPhyEnumerationPrinter (oldState)
00064:                 << " to " << LrWpanHelper::LrWpanPhyEnumerationPrinter (newState));
00065: */
00066: }
00067:
00068: int main (int argc, char *argv[])
00069: {
00070:   bool verbose        = false;
00071:   bool extended       = false;
00072:   uint32_t NOD        = 1000;
00073:   uint32_t Seed       = 1;
00074:
00075:   CommandLine cmd;
00076:   cmd.AddValue ("verbose", "turn on all log components", verbose);
00077:   cmd.AddValue ("extended", "use extended addressing", extended);
00078:   cmd.AddValue ("NOD", "The number of meter devices", NOD);
00079:   cmd.AddValue ("Seed", "Seed number", Seed);
00080:   cmd.Parse (argc, argv);
00081:
```

```
00082:   SeedManager::SetSeed (Seed);
00083:   LrWpanHelper lrWpanHelper;
00084:   if (verbose)
00085:     {
00086:       lrWpanHelper.EnableLogComponents ();
00087:     }
00088:
00089:   // Enable calculation of FCS in the trailers. Only necessary when interacting with real devices or wireshark.
00090:   // GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
00091:
00092:   // Create 2 nodes, and a NetDevice for each one
00093:   NodeContainer nodes;
00094:   nodes.Create(NOD);
00095:   //NS_LOG_UNCOND ("end of device configuration1");
00096:
00097:   std::vector<Ptr<LrWpanNetDevice>> dev;
00098:   std::vector<Ptr<LrWpanNetDevice>>::iterator it;
00099:   dev.reserve(NOD);
00100:   it=dev.begin();
00101:
00102:   for (uint32_t i=0; i<NOD; i++)
00103:     {
00104:        dev.insert(it, CreateObject<LrWpanNetDevice> ());
00105:     }
00106:
00107:   if (!extended)
00108:      {
00109:        for (uint32_t i=0; i<NOD; i++)
00110:         {
00111:            dev.at(i)->SetAddress (Mac16Address::Allocate ());
00112:            std::cout << dev.at(i)->GetMac()->GetShortAddress() << std::endl;
00113:         }
00114:      }
00115:    else
00116:      {
00117:        for (uint32_t i=0; i<NOD ;i++)
00118:         {
00119:            dev.at(i)->GetMac()->SetExtendedAddress(Mac64Address::Allocate ());
00120:            std::cout << dev.at(i)->GetMac()->GetExtendedAddress() << std::endl;
00121:         }
00122:      }
00123:
00124:   // Each device must be attached to the same channel
00125:   Ptr<SingleModelSpectrumChannel> channel = CreateObject<SingleModelSpectrumChannel> ();
00126:   Ptr<LogDistancePropagationLossModel> propModel = CreateObject<LogDistancePropagationLossModel> ();
00127:   Ptr<ConstantSpeedPropagationDelayModel> delayModel = CreateObject<ConstantSpeedPropagationDelayModel
> ();
00128:   channel->AddPropagationLossModel (propModel);
00129:   channel->SetPropagationDelayModel (delayModel);
00130:
00131:   for (uint32_t i=0; i<NOD; i++)
00132:     {
00133:        dev.at(i)->SetChannel (channel);
00134:     }
00135:
00136:   // To complete configuration, a LrWpanNetDevice must be added to a node
00137:   for (uint32_t i=0; i<NOD; i++)
00138:     {
00139:        nodes.Get(i)->AddDevice(dev.at(i));
00140:     }
00141:
00142:   // Trace state changes in the phy
00143:   dev.at(0)->GetPhy ()->TraceConnect ("TrxState", std::string ("phy0"), MakeCallback (&
00143: StateChangeNotification));
00144:
00145:   //box size per 1 device
00146:   //double x = 192.5/1000;
00147:   //double y = 180/1000;
00148:   //double z = 102.5/1000;
00149:   MobilityHelper mobility;
00150:   mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
00151:                                  "MinX", DoubleValue (100),
00152:                                  "MinY", DoubleValue (100),
00153:                                  "DeltaX", DoubleValue (192.5/1000),
00154:                                  "DeltaY", DoubleValue (180.0/1000),
00155:                                  "Z", DoubleValue (102.5/1000),
00156:                                  "GridWidth", UintegerValue (100),
00157:                                  "LayoutType", StringValue ("RowFirst"));
00158:    // each object will be attached a static position.
00159:    // i.e., once set by the "position allocator", the
00160:    // position will never change.
00161:    mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
00162:
00163:    // finalize the setup by attaching to each object
00164:    // in the input array a position and initializing
00165:    // this position with the calculated coordinates.
00166:    mobility.Install (nodes);
00167:
00168:    // iterate our nodes and print their position.
```

```
00169:    for (NodeContainer::Iterator j = nodes.Begin ();
00170:         j != nodes.End (); ++j)
00171:      {
00172:        Ptr<Node> object = *j;
00173:        Ptr<MobilityModel> position = object->GetObject<MobilityModel> ();
00174:        NS_ASSERT (position != 0);
00175:        Vector pos = position->GetPosition ();
00176:        std::cout << "x=" << pos.x << ", y=" << pos.y << ", z=" << pos.z << std::endl;
00177:      }
00178:
00179:   //Reception packet count of gateway
00180:   McpsDataConfirmCallback cb0;
00181:   cb0 = MakeCallback (&DataConfirm);
00182:   dev[0]->GetMac ()->SetMcpsDataConfirmCallback (cb0);
00183:   McpsDataIndicationCallback cb1;
00184:   cb1 = MakeCallback (&DataIndication);
00185:   dev[0]->GetMac ()->SetMcpsDataIndicationCallback (cb1);
00186:
00187:   // Tracing
00188:   lrWpanHelper.EnablePcapAll (std::string ("logdata/lr-wpan-data"), true);
00189:   AsciiTraceHelper ascii;
00190:   Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("lr-wpan-data.tr");
00191:   lrWpanHelper.EnableAsciiAll (stream);
00192:
00193:   // The below should trigger two callbacks when end-to-end data is working
00194:   // 1) DataConfirm callback is called
00195:   // 2) DataIndication callback is called with value of 50
00196:   //params.m_txOptions = TX_OPTION_ACK;
00197:
00198:   double min = 0.0;
00199:   double max = 3*3600;
00200:   Ptr<UniformRandomVariable> startTimeSeed = CreateObject<UniformRandomVariable> ();
00201:   startTimeSeed->SetAttribute ("Min", DoubleValue (min));
00202:   startTimeSeed->SetAttribute ("Max", DoubleValue (max));
00203:
00204:   for (uint32_t i=1; i<NOD; i++)
00205:     {
00206:       Ptr<Packet> p0 = Create<Packet> (50);  // 50 bytes of dummy data
00207:       McpsDataRequestParams params;
00208:       params.m_dstPanId = 0;
00209:       params.m_srcAddrMode = EXT_ADDR;
00210:       params.m_dstAddrMode = EXT_ADDR;
00211:       params.m_dstExtAddr = dev.at(0)->GetMac ()->GetExtendedAddress();
00212:       params.m_msduHandle = 0;
00213:       //std::cout << uint32_t(startTimeSeed->GetValue()) << std::endl;
00214:       Simulator::ScheduleWithContext (1, Seconds (uint32_t(startTimeSeed->GetValue())),
00215:                                       &LrWpanMac::McpsDataRequest,
00216:                                       dev.at(i)->GetMac (), params, p0);
00217:     }
00218:
00219:   std::cout << NOD << " device and # of device per H,W,L : " << cbrt(NOD) << " : " << ceil(cbrt(NOD))
00219:  << std::endl;
00220:   Simulator::Run ();
00221:   std::cout << RECV_STATIC << std::endl;
00222:   Simulator::Destroy ();
00223:   return 0;
00224: } ? end main ?
```