```
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:
00038: #include <iostream>
00039:
00040: using namespace ns3;
00041:
00042: static void DataIndication (McpsDataIndicationParams params, Ptr<Packet> p)
00043: {
00044:   NS_LOG_UNCOND ("Received packet of size " << p->GetSize ());
00045: }
00046:
00047: static void DataConfirm (McpsDataConfirmParams params)
00048: {
00049:   NS_LOG_UNCOND ("LrWpanMcpsDataConfirmStatus = " << params.m_status);
00050: }
00051:
00052: static void StateChangeNotification (std::string context, Time now, LrWpanPhyEnumeration
00052: oldState, LrWpanPhyEnumeration newState)
00053: {
00054:   NS_LOG_UNCOND (context << " state change at " << now.GetSeconds ()
00055:                          << " from " << LrWpanHelper::LrWpanPhyEnumerationPrinter (oldState)
00056:                          << " to " << LrWpanHelper::LrWpanPhyEnumerationPrinter (newState));
00057: }
00058:
00059: int main (int argc, char *argv[])
00060: {
00061:   bool verbose = false;
00062:   bool extended = false;
00063:
00064:   CommandLine cmd;
00065:
00066:   cmd.AddValue ("verbose", "turn on all log components", verbose);
00067:   cmd.AddValue ("extended", "use extended addressing", extended);
00068:
00069:   cmd.Parse (argc, argv);
00070:
00071:   LrWpanHelper lrWpanHelper;
00072:   if (verbose)
00073:     {
00074:       lrWpanHelper.EnableLogComponents ();
00075:     }
00076:
00077:   // Enable calculation of FCS in the trailers. Only necessary when interacting with real devices or wireshark.
00078:   // GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
00079:
00080:   // Create 2 nodes, and a NetDevice for each one
00081:   Ptr<Node> n0 = CreateObject <Node> ();
00082:   Ptr<Node> n1 = CreateObject <Node> ();
00083:
```

```
00084:   Ptr<LrWpanNetDevice> dev0 = CreateObject<LrWpanNetDevice> ();
00085:   Ptr<LrWpanNetDevice> dev1 = CreateObject<LrWpanNetDevice> ();
00086:
00087:   if (!extended)
00088:     {
00089:       dev0->SetAddress (Mac16Address ("00:01"));
00090:       dev1->SetAddress (Mac16Address ("00:02"));
00091:     }
00092:   else
00093:     {
00094:       Ptr<LrWpanMac> mac0 = dev0->GetMac();
00095:       Ptr<LrWpanMac> mac1 = dev1->GetMac();
00096:       mac0->SetExtendedAddress (Mac64Address ("00:00:00:00:00:00:00:01"));
00097:       mac1->SetExtendedAddress (Mac64Address ("00:00:00:00:00:00:00:02"));
00098:     }
00099:
00100:   // Each device must be attached to the same channel
00101:   Ptr<SingleModelSpectrumChannel> channel = CreateObject<SingleModelSpectrumChannel> ();
00102:   Ptr<LogDistancePropagationLossModel> propModel = CreateObject<LogDistancePropagationLossModel> ();
00103:   Ptr<ConstantSpeedPropagationDelayModel> delayModel = CreateObject<ConstantSpeedPropagationDelayModel
> ();
00104:   channel->AddPropagationLossModel (propModel);
00105:   channel->SetPropagationDelayModel (delayModel);
00106:
00107:   dev0->SetChannel (channel);
00108:   dev1->SetChannel (channel);
00109:
00110:   // To complete configuration, a LrWpanNetDevice must be added to a node
00111:   n0->AddDevice (dev0);
00112:   n1->AddDevice (dev1);
00113:
00114:   // Trace state changes in the phy
00115:   dev0->GetPhy ()->TraceConnect ("TrxState", std::string ("phy0"), MakeCallback (&
00115: StateChangeNotification));
00116:   dev1->GetPhy ()->TraceConnect ("TrxState", std::string ("phy1"), MakeCallback (&
00116: StateChangeNotification));
00117:
00118:   Ptr<ConstantPositionMobilityModel> sender0Mobility = CreateObject<ConstantPositionMobilityModel> ();
00119:   sender0Mobility->SetPosition (Vector (0,0,0));
00120:   dev0->GetPhy ()->SetMobility (sender0Mobility);
00121:   Ptr<ConstantPositionMobilityModel> sender1Mobility = CreateObject<ConstantPositionMobilityModel> ();
00122:   // Configure position 10 m distance
00123:   sender1Mobility->SetPosition (Vector (0,10,0));
00124:   dev1->GetPhy ()->SetMobility (sender1Mobility);
00125:
00126:   McpsDataConfirmCallback cb0;
00127:   cb0 = MakeCallback (&DataConfirm);
00128:   dev0->GetMac ()->SetMcpsDataConfirmCallback (cb0);
00129:
00130:   McpsDataIndicationCallback cb1;
00131:   cb1 = MakeCallback (&DataIndication);
00132:   dev0->GetMac ()->SetMcpsDataIndicationCallback (cb1);
00133:
00134:   McpsDataConfirmCallback cb2;
00135:   cb2 = MakeCallback (&DataConfirm);
00136:   dev1->GetMac ()->SetMcpsDataConfirmCallback (cb2);
00137:
00138:   McpsDataIndicationCallback cb3;
00139:   cb3 = MakeCallback (&DataIndication);
00140:   dev1->GetMac ()->SetMcpsDataIndicationCallback (cb3);
00141:
00142:   // Tracing
00143:   lrWpanHelper.EnablePcapAll (std::string ("lr-wpan-data"), true);
00144:   AsciiTraceHelper ascii;
00145:   Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("lr-wpan-data.tr");
00146:   lrWpanHelper.EnableAsciiAll (stream);
00147:
00148:   // The below should trigger two callbacks when end-to-end data is working
00149:   // 1) DataConfirm callback is called
00150:   // 2) DataIndication callback is called with value of 50
00151:   Ptr<Packet> p0 = Create<Packet> (50);  // 50 bytes of dummy data
00152:   McpsDataRequestParams params;
00153:   params.m_dstPanId = 0;
00154:   if (!extended)
00155:     {
00156:       params.m_srcAddrMode = SHORT_ADDR;
00157:       params.m_dstAddrMode = SHORT_ADDR;
00158:       params.m_dstAddr = Mac16Address ("00:02");
00159:     }
00160:   else
00161:     {
00162:       params.m_srcAddrMode = EXT_ADDR;
00163:       params.m_dstAddrMode = EXT_ADDR;
00164:       params.m_dstExtAddr = Mac64Address ("00:00:00:00:00:00:00:02");
00165:     }
00166:   params.m_msduHandle = 0;
00167:   params.m_txOptions = TX_OPTION_ACK;
00168: //  dev0->GetMac ()->McpsDataRequest (params, p0);
```

```
00169:   Simulator::ScheduleWithContext (1, Seconds (0.0),
00170:                                    &LrWpanMac::McpsDataRequest,
00171:                                    dev0->GetMac (), params, p0);
00172:
00173:   // Send a packet back at time 2 seconds
00174:   Ptr<Packet> p2 = Create<Packet> (60);  // 60 bytes of dummy data
00175:   if (!extended)
00176:     {
00177:       params.m_dstAddr = Mac16Address ("00:01");
00178:     }
00179:   else
00180:     {
00181:       params.m_dstExtAddr = Mac64Address ("00:00:00:00:00:00:00:01");
00182:     }
00183:   Simulator::ScheduleWithContext (2, Seconds (2.0),
00184:                                    &LrWpanMac::McpsDataRequest,
00185:                                    dev1->GetMac (), params, p2);
00186:
00187:   Simulator::Run ();
00188:
00189:   Simulator::Destroy ();
00190:   return 0;
00191: } ? end main ?
```