ISA 100.11a ns3 Developers Guide

Geoffrey Messier
FISH Lab
University of Calgary

1 Introduction

This document describes the Fully Integrated Systems and Hardware (FISH) Lab implementation of an ns3 simulation of the ISA 100.11a standard.

2 Using Doxygen

This document is meant to provide a high level overview of how the code works. Once reading it, you will need to dive into the code to get a full understanding of how the simulator works. To help you navigate the code, the source code has been documented to be compliant with doxygen. Doxygen is a tool that parses the comments in a source code directory and generates a website based on those comments and the overall code structure.

2.1 Using Doxygen

The SVN repository trunk for the ISA code contains the Doxyfile configuration file and the `doxygen` directory that will contain the html pages. That directory is empty since there's no point storing doxygen output on the SVN server. To generate the doxygen page, install doxygen on your machine, cd into the ISA directory and type `doxygen`.

3 The Standards

This simulation is based on two standards: ISA100.11a and 802.15.4 [1, 2]. The ISA standard defines everything from the data link layer up and is built on top of an 802.15.4 physical layer. The documents for the current ISA standard and the 2006 802.15.4 standard are contained in the `doc` directory. The 802.15.4 standard is a bit out of date but that's the document that the `LrWpan` project bases it's code on and we use the `LrWpan` PHY layer in our simulation. However, we have renamed the `LrWpan` code to `ZigbeePhy` to differentiate it from the `LrWpan` project. The comments refer to specific tables and section numbers in the standard so it's important to have the exact version of the 802.15.4 document.

4 Example Simulations

In the `examples` directory in the ISA repository, you will find a number of simulations but the one you should concentrate on first is:

- `isa100-three-node-example.cc` A simple simulation where two nodes send data to a third. Demonstrates the use of the application class and the superframe schedule. Also demonstrates tracing.
- `isa100-source-route-test.cc` Demonstrates how source routing can be used to deliver packets over a multi-hop link.
- `isa100-link-test.cc` This simulation demonstrates the use of the channel model and simulates a link for a variety of channel attenuations to demonstrate how packet loss is tracked.
- `isa100-random-net.cc` This larger simulation demonstrates how to simulate a number of random network topologies, run a separate simulation for each one and record performance statistics. This is the simulation used to generate the results in [3]. Note that unlike the other examples, this simulation takes command line arguments to allow a linux shell script to call the simulator many times for different parameters. For example, running the simulator for its first iteration (iteration 0), 20 sensor nodes, a random number seed of 10 and packet level convex integer optimization, you would type

```
./waf  --run scratch/isa100-random-net \
  --command-template="%s -optType=ConvIntPckt -iter=0 \
                      -rndSeed=10 -nnodes=20"
```

5 Software Architecture

Each node in the simulation is implemented as a `Node` object. This object hosts other objects that implement the different parts of the simulated protocol. Nodes are created and these objects are installed within each node object using routines within the Isa100Helper class.
Each node hosts an Isa100Application and an Isa100NetDevice object. The Isa100NetDevice object contains an number of other objects that implement the protocol stack functionality and possibly other node hardware components (sensors, processors, etc.). Currently, two protocol stack layers are implemented: the physical layer in the ZigbeePhy class and the data link layer in the Isa100Dl class. The role of the Isa100Application object is to generate data addressed to a specific destination and feed it to the Isa100NetDevice which looks after delivering the data to that destination. Applications can also receive data. The generation and reception of data sometimes involves interacting with a hardware object (sensor, actuator, etc.).

5.1 Isa100Application

A derived class of `ns3::Application`, this is the class that generates packets for transmission and also processes received packets. When packets are generated, they are passed to a Isa100Dl data link layer protocol object for transmission via the callback function stored in `m_dlDataRequest`. This callback function pointer typically points to the `Isa100Dl::DlDataRequest` function. When packets are received, the Isa100Dl object passes them to the application layer by calling the `Isa100Application::DlDataIndication()` virtual function. This function is virtual since it allows us to create different applications as derived classes of Isa100Application. These derived classes are described below. The parameters for generating the packets are controlled by the `Isa100Application::SetAttribute` function.

5.1.1 Isa100FieldNodeApplication

This application is for sensor nodes generating data by sampling a sensor and then transmitting that data to a central access point via the multi-hop network. This application has pointers to both the sensor and processor objects contained within the node. Each sensing operation is triggered using the `StartSensing()` function. This function turns on the processor, asks the sensor to start sensing and then schedules the next sensing operation before terminating. Note that a packet with sensor data is not actually generated until the sensing operation is complete. When this occurs, the sensor calls the `SensorSampleCallback()` function via a callback function pointer. `SensorSampleCallback()` simply generates the packet and passes it to the DL object.

5.1.2 Isa100BackboneNodeApplication

This is a very simple sink node application that simply receives packets inside the `DlDataIndication()` function and notes the receiption by generating a logfile message.

5.2 Isa100Helper

This class assists in setting up the simulation by hiding some of the complexity of configuring the network objects. While there is only one Isa100Helper class, its functionality is divided across three source files as follows:

- `isa100-helper.cc`: This file contains functions that connect and configure the different objects contained inside Isa100NetDevice that implement the physical layer, the data link layer, battery, processor and sensor.
- `isa100-helper-locations.cc`: This file contains the functions used to generate and assign random node positions.
- `isa100-helper-scheduling.cc`: This simulation library is able to draw on several optimization algorithms (minimum hop, network lifetime maximization, etc.) to determine traffic routes through a multihop network. All of these algorithms produce solutions in the form of how many packets should flow from one node to the next. This file contains the functions that then map these flow solutions into generate TDMA schedules and source routing lists for each node in the network to implement this information flow.

  The routine `Isa100Helper::CreateOptimizedTdmaSchedule()` is the master function that is called within the main ns3 simulation program that generates the optimal routes. It selects the routing algorithm used to determine the packet flow matrix for the network. It then calls `Isa100Helper::ScheduleAndRouteTdma()` to translate the flow matrix to the TDMA schedule and source routing lists as described in [3].

5.3 Network Optimization

Optimization is used within Isa100Helper to determine the matrix of packet flows through the multi-hop network. All optimization routes are derived classes of the base class `TdmaOptimizerBase` which implements some of the common setup operations required by all the optimization routines. This consists primarily of determining the energy cost of transmitting over the different links. The `MinHopTdmaOptimizer` class determines packet flow using a variant of the breadth first search algorithm that finds the paths that minimize the number of hops required by

each packet to reach the sink node. The `GoldsmithTdmaOptimizer` class solves the flow matrix by using a convex optimization to maximize network lifetime as described in [3, 4]. Finally, `ConvexIntTdmaOptimizer` uses a convex integer optimization to maximize network lifetime but, unlike `GoldsmithTdmaOptimizer`, it produces superior results by working in units of packets rather than bits [3].

5.4 Isa100NetDevice

This class represents the physical node and contains the objects that implement different node functionality including the physical layer, data link layer, battery, processor and sensor. It also contains the functions that connect the callback routines that allow the physical and data link layers to communicate. It's necessary to connect the callback functions within the net device since only the net device is aware when both the physical layer and data link layer objects have been installed.

5.5 Isa100Battery

This object keeps track of how much energy the node has and has the capability via the callback function `m_depletionCallback` to terminate the simulation when a node runs out of energy. All objects that consume energy (ie. the physical layer, the sensor, the processor, etc.) call the battery function `Isa100Battery::DecrementEnergy()` each time they perform an operation that consumes energy. The object calling `Isa100Battery::DecrementEnergy()` also passes in a string indicating what time of operation is consuming the energy. The Isa100Battery object keeps track of the energy consumed by each operation category and can print an energy breakdown to a logfile.

5.6 Isa100Sensor

The Isa100Sensor object models the operation of the node sensor. A call to the `Isa100Sensor::StartSensing()` begins the sensing operation which automatically schedules a call to `Isa100Sensor::EndSensing()`. The end sensing operation will decrement the battery energy and will notify the application of the sensing operation completion via the `m_sensingCallback` callback function.

5.7 Isa100Processor

The Isa100Processor object is a simple class that allows an application to switch its status between two states: sleeping and awake. Each time the processor changes its state, it decrements the appropriate amount of energy from the battery object through the use of battery callback functions.

5.8 Isa100Dl

As discussed in Section 5.3.4 of [1], a data link (DL) subnet encompases the wireless mesh part of an ISA 100 network. Therefore, in addition to medium access control, the DL is also where the mesh routing is implemented. ISA 100 does have a network layer but it exists mainly to route between different DL subnets.

Note: The ARQ mechanism in the DL code is currently under development. It should be considered broken at this time.

5.8.1 Medium Access Control

The `Isa100Dl` makes use of a `Isa100DlSfSchedule` helper class that is used to store the hopping and link activity schedule that indicates when the node can transmit/receive during the superframe. Due to the way attributes are implemented in ns3, a helper class was necessary to allow the schedule to be programmed into the DL object via `Isa100Dl::SetAttribute`.
One of the primary functions of `Isa100Dl` is to implement the channel hopping and superframe timeslot link activity scheduling as described in [1]. This is accomplished by the `Isa100Dl::ChannelHop()` and `Isa100Dl::ProcessLink()` functions. `Isa100Dl::ChannelHop()` is scheduled to execute and change the channel at the start of each superframe time slot. `Isa100Dl::ProcessLink()` executes at the start of each time slot where the node is scheduled to be active and does the following:

- Turn the receiver on if the link is RECEIVE or if the link is SHARED and there are no packets to send.
- Turn the transmitter on if the link is TRANSMIT and there's a packet to send. Note that function only makes a request to the `ZigbeePhy` object to change the state to TRANSMIT_ON. When `ZigbeePhy` does turn on the transmitter, it lets the DL object know by calling `Isa100Dl::PlmeSetTrxStateConfirm`. It's within this member function that the packet is actually transmitted by making a call to `LrWpanPhy::PdDataRequest`.
- Request a channel clear assessment (CCA) if the link is shared, there's a packet to transmit and the node is not in backoff mode. Backoff mode is when the node delays transmission if it senses the channel is busy when in SHARED mode. When the `ZigbeePhy` object finishes the CCA request, it calls `Isa100Dl::PlmeCcaConfirm` which turns the transmitter on if the channel is idle. If it's busy, the DL object goes into backoff mode by setting the `m_expBackoffCounter` variable.

When a packet is submitted to the DL for transmission via the `Isa100Dl:DlDataRequest` function, the DL object just prepares the packet header and puts it in the queue for transmission. The packet will be sent in the next timeslot identified as TRANSMIT or SHARED and idle. When a packet is received by the `ZigbeePhy` object, it passes it to the DL via the `Isa100Dl::PdDataIndication` function. This function checks to see if the packet is addressed to the current node. If so, it passes it up to the upper layers of the protocol stack.

5.8.2 Routing

Routing is implemented as a based class called `Isa100RoutingAlgorithm` stored within `Isa100Dl`. Specific routing algorithms are implemented as derived classes from `Isa100RoutingAlgorithm`. When a packet is transmitted, the `Isa100Dl::DlDataRequest` routine calles `Isa100RoutingAlgorithm::PrepTxPacketHeader` which adds the additional information to the packet header that the routing algorithm requires. When a packet is received within `Isa100Dl::ProcessPdDataIndication`, the packet is passed to `Isa100RoutingAlgorithm::ProcessRxPacket` which determines whether the packet is at its final destination or needs to be forwarded on.

The only routing algorithm currently implemented is *source routing* which is where the sending node specifies the entire multi-hop route the packet must take to its destination. It is assumed that the source is given global knowledge of the network topology during a startup phase. The source routing derived class is `Isa100SourceRoutingAlgorithm`. The packet fields required by the source routing class are specified in `Isa100DlHeader`.

5.9 ZigbeePhy

Since the ISA100.11a standard is designed to operate on top of a standard 802.15.4 PHY, the code used to implement the PHY in this simulation is borrowed from the LrWpan project. However, the `LrWpanPhy` has been extensively rewritten to work properly with this ISA implementation so the main stream LrWpan project code will no longer work with the ISA simulation. The modified code has been renamed `ZigbeePhy` and it resides in the ISA100 directory.
The `ZigbeePhy` code uses the `SpectrumPhy` code to actually communicate between nodes. The `SpectrumPhy` code used is the standard ns3 release and is not stored in the ISA FISH repository.
The `ZigbeePhy` object is concerned mainly with modelling the state changes of the transceiver (ie. TX_ON, RX_ON, RX_BUSY, etc.) and modelling the finite time required to send and receive a packet.
On the receive side, `ZigbeePhy::StartRx()` is called by the lower level channel object whenever a signal arrives that is strong enough for the node to pick up. If the PHY object is in RX_ON, it transitions to RX_BUSY and schedules `ZigbeePhy::EndRx()` to execute at the end of the packet. If the PHY object is already in RX_BUSY, that means a second packet has arrived while the first one was being received. This is a collision. The new packet is dropped and the existing packet is marked as corrupt. When `ZigbeePhy::EndRx()` executes, it passes the packet up to the `Isa100Dl` object as long as it hasn't been marked as corrupt.
One the transmit side, `ZigbeePhy:PdDataRequest` is called when a packet is given to `ZigbeePhy` for transmission and `ZigbeePhy::EndTx()` is then scheduled to execute at the end of the packet duration.
The `ZigbeePhy` object also performs CCA by either looking at received energy levels or the number of 802.15.4 signals being received. The number of received signals is a simple counter that is incremented by each call to `ZigbeePhy::StartRx` and decremented by `ZigbeePhy::DecrementChannelRxSignals` when the packet finishes. The received energy level used for the CCA is also updated by `ZigbeePhy::StartRx`.

5.9.1 SpectrumPhy

The `ZigbeePhy` code is a derived class from the SpectrumPhy class which is part of a generalized spectrum aware `spectrum` physical layer code specified in [5]. The purpose of this code is to model a frequency selective channel as a series of bands. This is useful for evaluating standards like ISA100.11a that make extensive use of frequency hopping.
The frequency bands are defined on a per-device basis and each band has a corresponding channel value that can be accessed and manipulated using the `SpectrumValue` class.
There are two important functions used for the channel/PHY interface. The first is `SpectrumChannel::StartTx` which takes a packet, a `SpectrumValue` instance, time duration and a reference to the envoking `SpectrumPhy` object. The `SpectrumPhy::StartRx` function is called by the channel on the receiving node and contains a reference to the received packet, the spectrum of the received signal, time duration and a reference to the sender PHY.

The `SpectrumPropagationLossModel` abstract base class holds the propagation model and takes a `SpectrumValue` object for its `CalcRxPower` function. It's also possible to define the spectral mask of the signal across frequencies.

5.9.2 Physical Layer Energy Consumption

Energy consumption of the physical layer is a function of transmit power. To allow for different energy consumption models, the current draw of the physical layer is implememnted in a separate `ZigbeeTrxCurrentModel` class. This model is initialized separately in the main simulation file and then stored inside the physical layer object using the `m_currentDraws` member pointer.

References

[1] *Wireless systems for industrial automation: Process control and related applications*, American National Standard Std. ANSI/ISA-100.11a-2011.

[2] I. C. Society, "IEEE std 802.15.4 - 2006," The Institute of Electrical and Electronics Engineers, Inc., 2006, new York, USA.

[3] M. J. Herrmann and G. G. Messier, "Cross-layer lifetime optimization for practical industrial wireless networks: A petroleum refinery case study," submitted to IEEE Transactions on Industrial Informatics, 2018.

[4] S. Cui, R. Madan, A. J. Goldsmith, and S. Lall, "Cross-layer energy and delay optimization in small-scale sensor networks," *IEEE Transactions on Wireless Communications*, vol. 6, no. 10, 2007.

[5] N. Baldo and M. Miozzo, "Spectrum-aware channel and phy layer modeling for ns3," in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS '09. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 2:1–2:8. [Online]. Available: http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7647