Federal Agency and Organization Element to Which Report is Submitted: 4900
Federal Grant or Other Identifying Number Assigned by Agency: 1816197

Project Title:

NeTS: Small: RUI: Bulldog Mote- Low Power Sensor Node and design Methodologies for Wireless Sensor Networks

Bulldog Mote Sensor Design

PD/PI Name:
Nan Wang, Principal Investigator
Woonki Na, Co-Principal Investigator: Lead Energy Harvesting Team

Recipient Organization:
California State University-Fresno Foundation

Team Members:
Calvin Jarrod Smith and Cameron Lane

Project/Grant Period:
10/01/2018 - 09/30/2021

Reporting Period:
10/01/2020 - 09/30/2021

Signature of Submitting Official (signature shall by submitted in accordance with agency specific instructions):

National Science Foundation Bulldog Mote Project

Progress as of October 1, 2021

Calvin Jarrod Smith
Cameron Lane

Supervisor: Dr. Nan Wang

Department of Electrical and Computer Engineering
California State University, Fresno
NSF NeTs Bulldog Mote Team

Contents

List of Figures

1 Purpose

The National Science Foundation (NSF) Bulldog Mote Team is part of California State University, Fresno Department of Electrical and Computer Engineering. The purpose of this group is to discover, research and develop wireless communication protocols and hardware used in Mobile Ad-Hoc Networks (MANETs) and Wireless Sensor Networks (WSNs). This team's purpose is to research current wireless technologies and algorithms and create new hardware implementations to support current and future protocols for both MANETs and WSNs.

To accomplish this goal, the team has researched new a developing wireless schemes, hardware to support the processing of data within these wireless networks and various routing algorithms used to direct data through MANETs and sensor networks.

2 Current Projects

The Bulldog Mote team is currently working on several sensor mote hardware implementations as well as researching and testing routing algorithms that help to lower energy consumption in a wireless sensor network or MANET. The current projects focus was to design wireless sensor devices that use low-power and can be run on battery alone. One project, still being developed, uses solar energy to keep the system running perpetually without the need for battery replacement or human involvement. The current and completed projects are listed below:

Updated NeTs Mote

An updated design of the first NeTs Mote, reported last year, was designed using different passive components to help in the testing and implementation.

SM14 Bulldog Mote Shield

The first step toward implementing a working wireless sensor device was to take an already existing development board, the SM14Z2538PA1, with a microcontroller that was to be used in future devices, and create a *shield* that fit over the development board's header pins. This shield contained various sensors and included two switching voltage regulators to power the digital and analog sensors separately as well as the microcontroller.

Bulldog Mote

The development of the Bulldog Mote came shortly after the SM14 Shield device. A custom PCB was designed for the entire system, as well as including additional power supply filters and external memory.

LS-AODV Routing Algorithm

A new routing algorithm was developed, Lightning Strike Ad-Hoc On-Demand Distance Vector (LS-AODV), to help increase the network lifetime in MANETs. The traditional AODV algorithm [1] was analyzed, including the Energy Balancing AODV (EB-AODV) [2]. Energy pathways were discovered during the analysis and the LS-AODV algorithm was conceived to take advantage of these pathways to distribute energy consumption throughout the entire network. This research was recently submitted to a conference publication.

3 Updated NeTs Mote

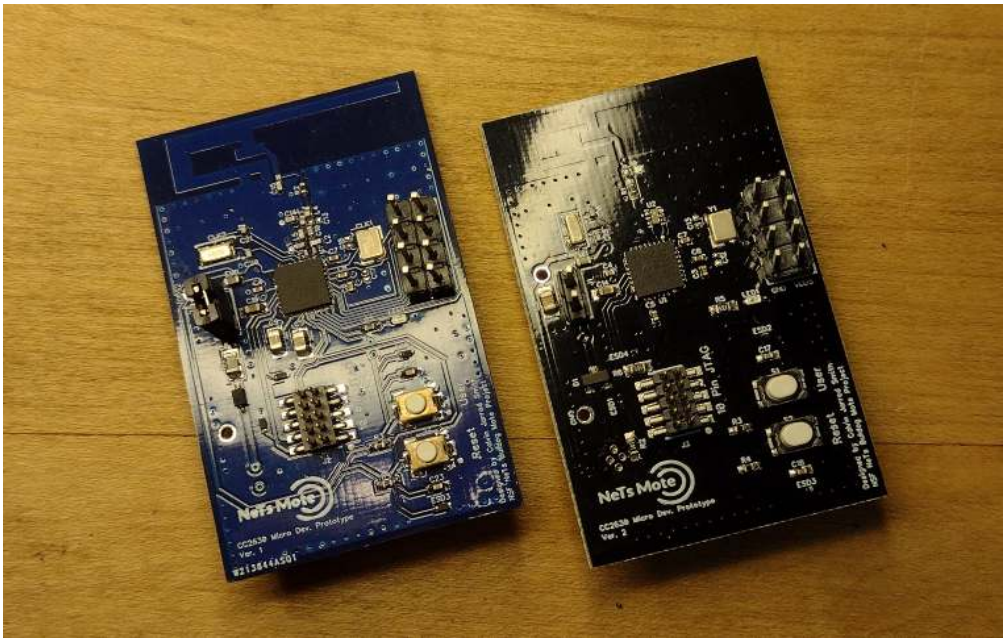


Figure 1: NeTs Mote (blue board) and Updated NeTs Mote (black board).

3.1 Overview

Last year, the NeTs Mote was designed and assembled using the TI CC2630 wireless microcontroller, custom PCB, inverted-F antenna, battery and various passive components. This system was designed using Texas Instrument's (TI) reference designs and chip recommendations. During testing the chip could not be recognized by the programming module, and was later found to have an error in the circuit design. This error was corrected as well as an integrated balun being included for the CC2630 chip. The new design showed a fully functioning antenna and transceiver circuit and demonstrated a fully functioning development board for the CC2630 micorcontroller.

3.2 Testing Previous NeTs Mote

It was observed after circuit analysis that the PCB layout did not have the correct supply voltage, VDDR, connection to the internal supply voltage, VDDS, as specified in the manufacturer datasheet. Therefore, no further testing was possible and programming could not be completed.

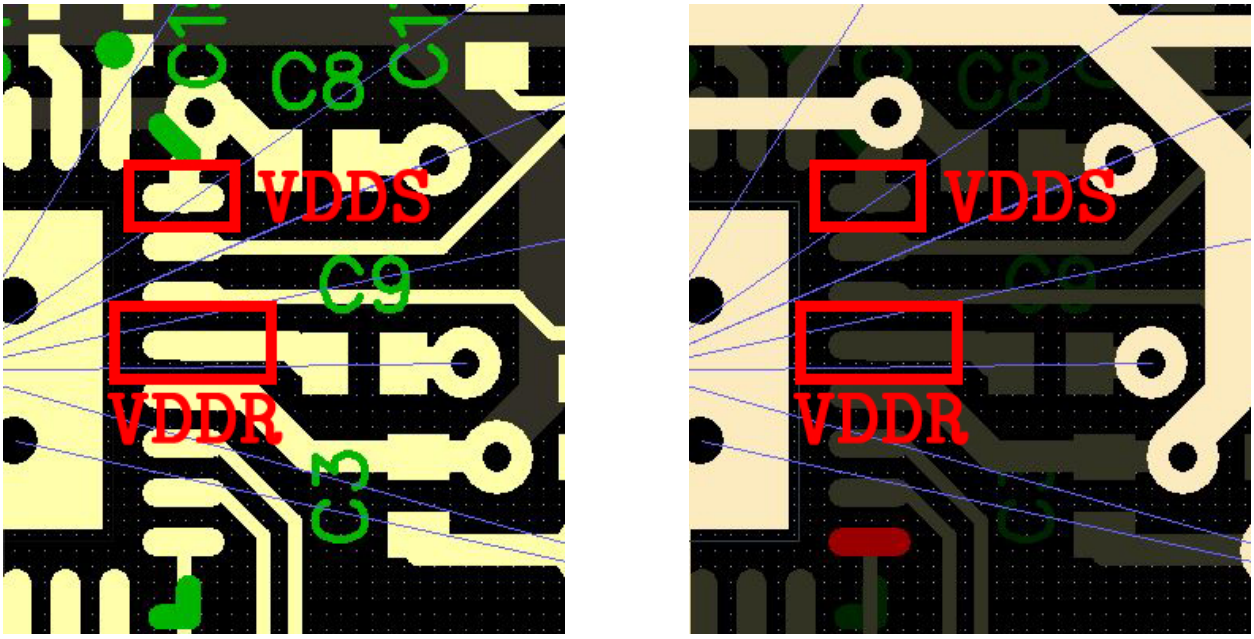


Figure 2: PCB Layout highlighting design error.

3.3 Circuit and PCB Design

In the new circuit and PCB design, the error with the supply voltages was corrected. In addition, the Balun used to filter and tune the signals to and from the antenna were replaced by an integrated Balun designed for the CC26XX line of TI chips. This made the assembly of the PCB significantly more streamlined. As well, the passive components were chosen to be a 0603 case size using imperial units, which also made placement significantly simpler compared to the 0201 component case size.

In the reference designs *Humidity and Temperature Sensor Node for Star Networks Enabling 10+ Year Coin Cell Battery Life* [3], the system contained a current limiting resistor in series with the battery. Although small, this resistor keeps current low in the system and prevents overdrawing the battery. If this were to happen, the voltage across the current limiting resistor would drop significantly enough to cause the CC2630 chip to enter a Brown Out state. For testing this system, the current limiting resistor was removed.

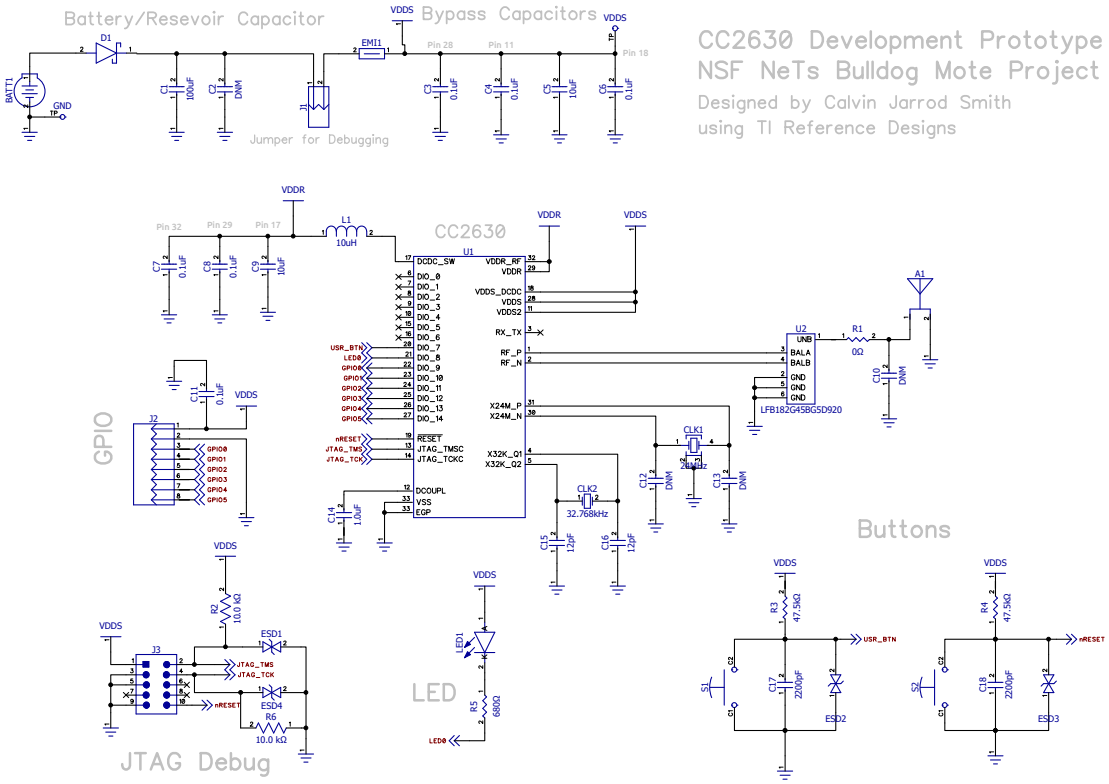


Figure 3: Circuit Layout of Updated NeTs Mote.

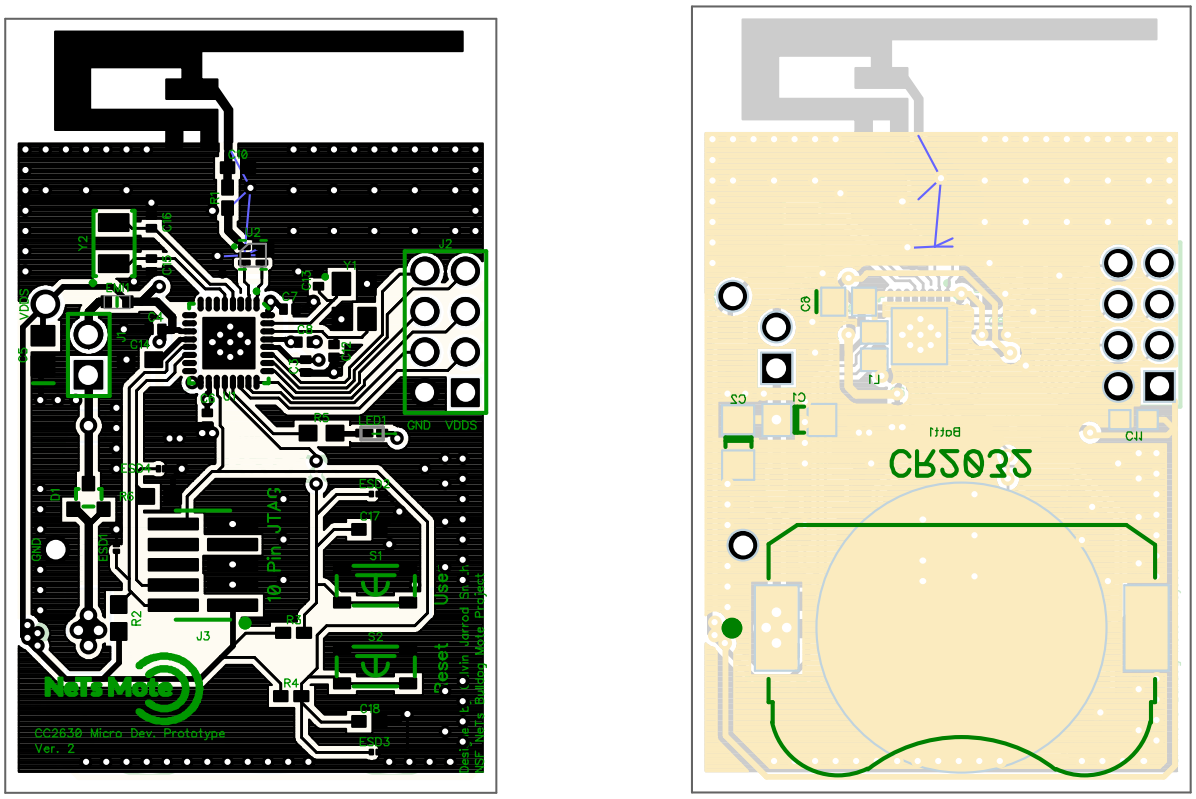


Figure 4: PCB Layout of Updated NeTs Mote.

3.4 Testing Procedure

For testing and programming, the system needed to be connected a 3 V power supply. The current supplied by a coin cell battery (which it was designed to operate with) is too small for flashing a program onto the chip. The test point for VDDS and GND were used for this purpose, bypassing the input diode and ferrite bead filter.

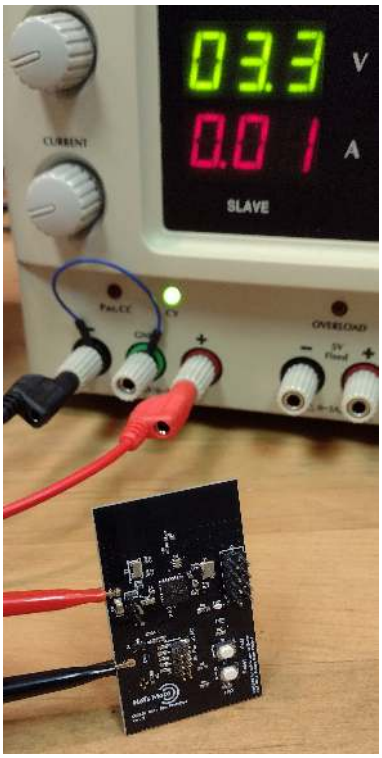


Figure 5: NeTs Mote Connected to External Power.

Then the JTAG was connected to the Olimex TMS320-XDS100v3 JTAG programmer and debugger. A program was written using the Contiki NG operating system to flash the on-board LED as a simple programmability test of the chip. This program was then flashed onto the CC2630 chip using TI Flash Programmer 2.

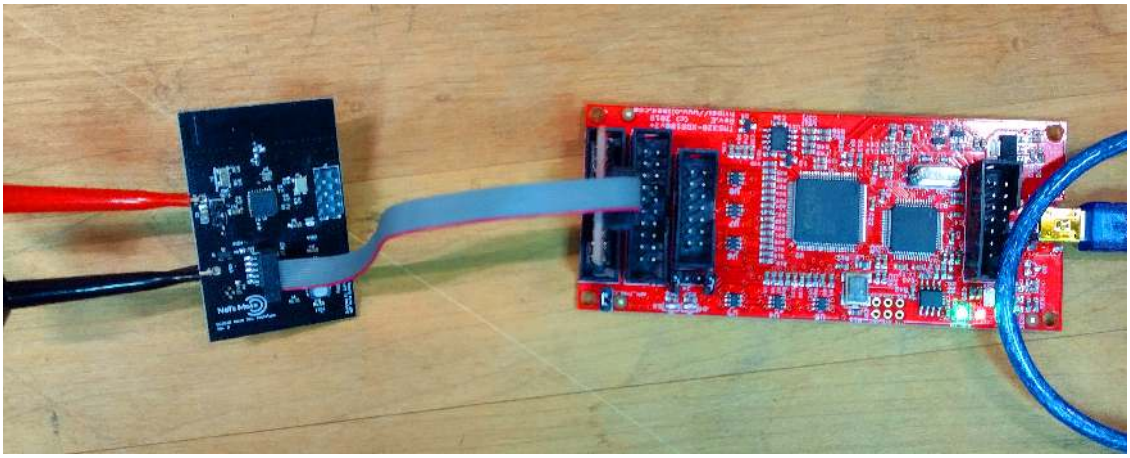


Figure 6: JTAG Connected to NeTs Mote.

The next step was to test the UART serial communication used by Contiki NG to debug their software. A USB to TTL adapter was used to connect the UART pins on the NeTs

mote to a host computer to read the serial messages. The software, *RealTerm*, was used to connect to the USB port the adapter was connected to display the output of the system.

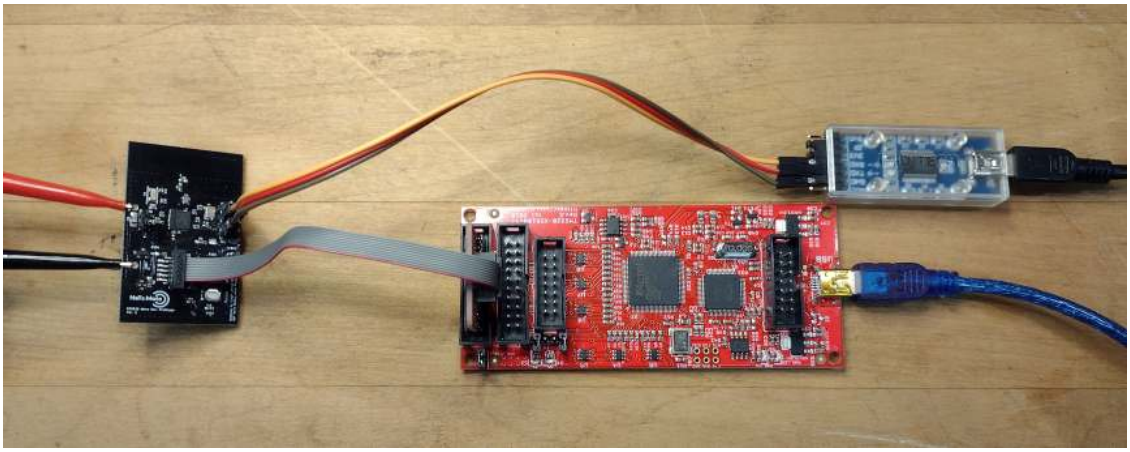


Figure 7: USB-to-TTL Adapter Connected to the UART Pins of the NeTs Mote.

The final step for testing was to ensure the antenna circuit and balun worked properly. To do this, the TI SmartRF Studio 7 software package was used, as well as a SmartRF06 Evaluation Board with CC2538EM daughter card attached to act as a receiver.

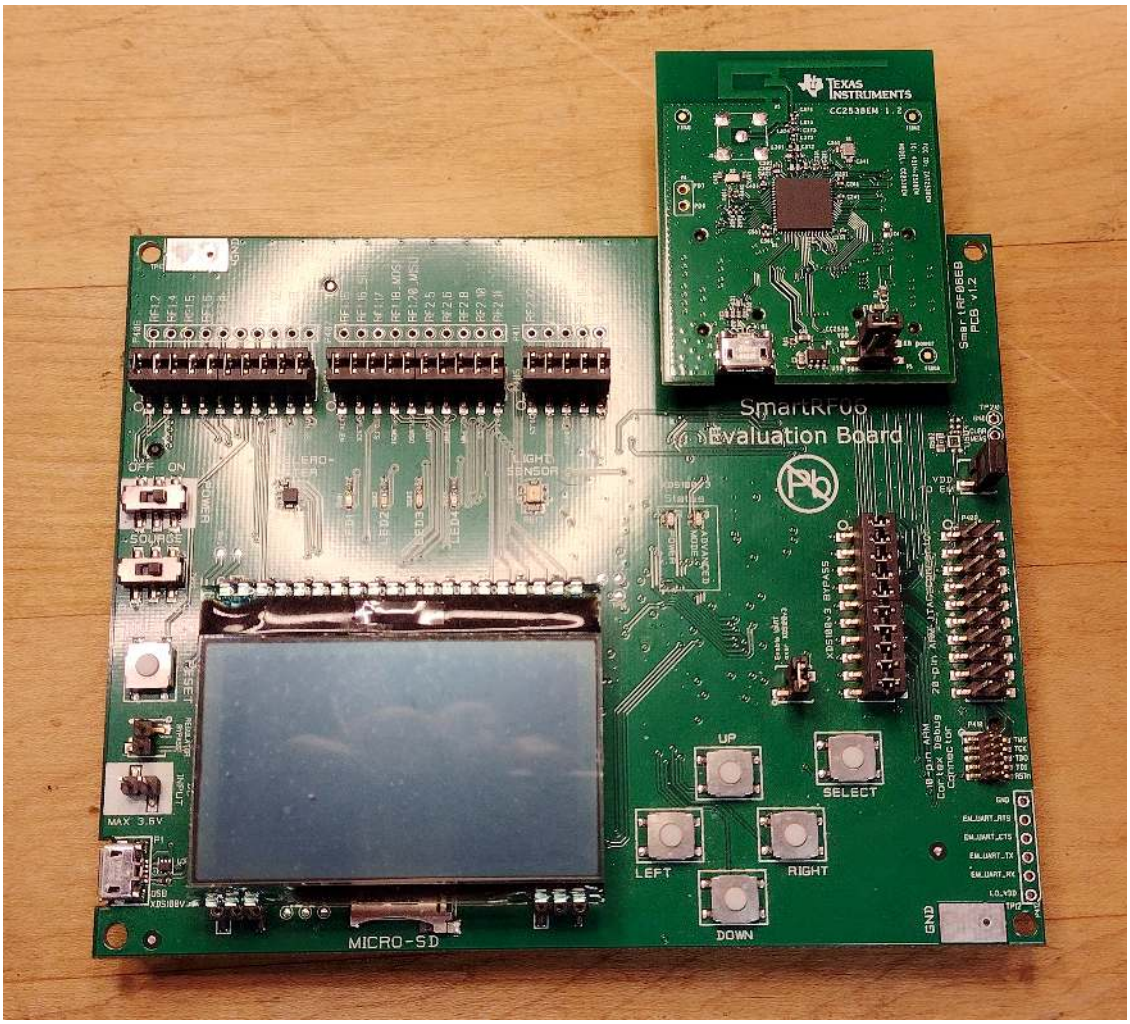


Figure 8: The SmartRF06 Evaluation Module used as a receiver.

The SmartRF06 board was placed about 2.4 m away from the device under test, with clear line-of-sight. This device was connected to a host computer as the receiver using the TI SmartRF Studio 7. Once plugged in, the CC2538 chip was selected out of the list of options.

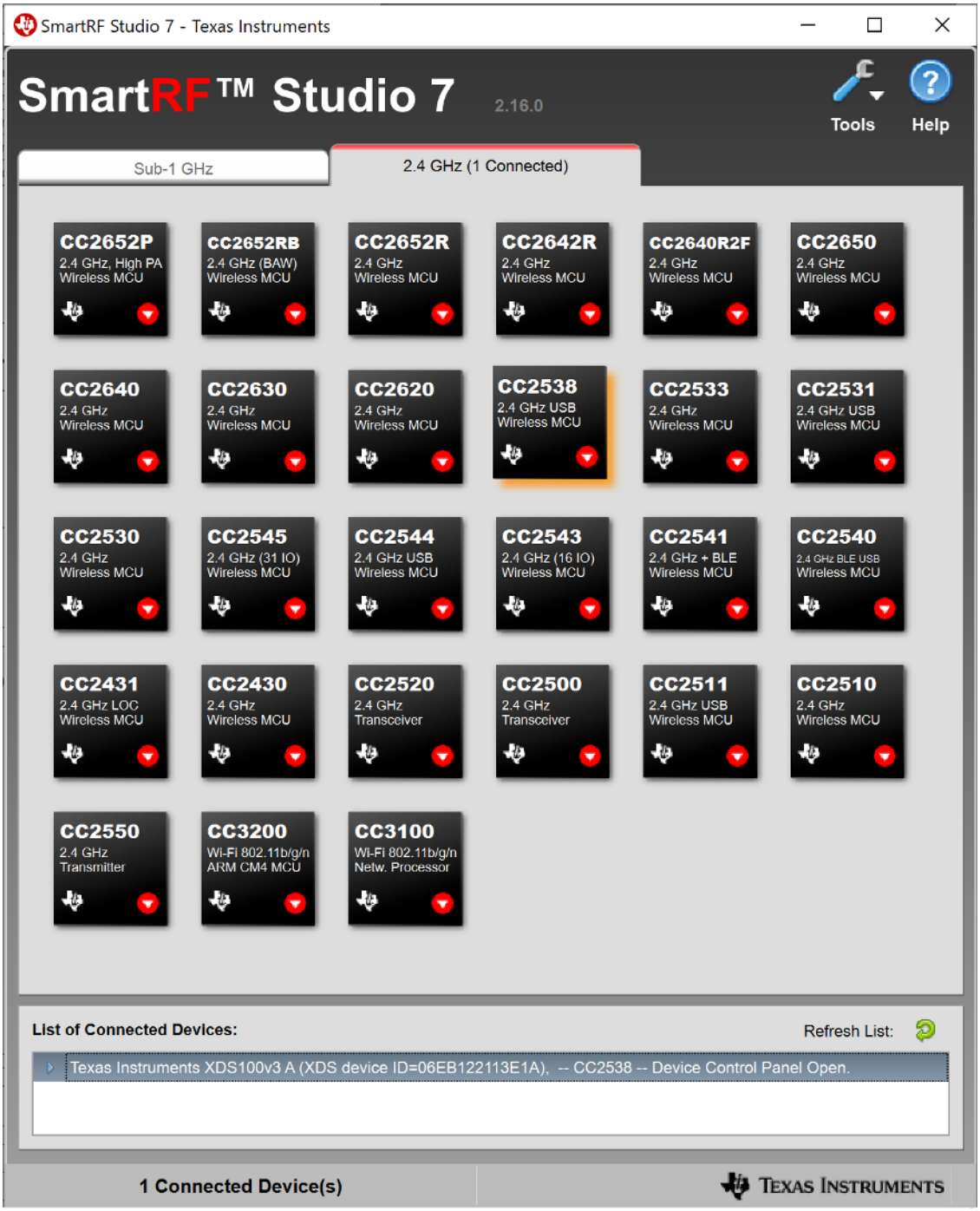


Figure 9: Selecting CC2538 chip in the SmartRF Studio.

Once this was selected, the *Simple RX* mode was selected and started. This idles any program active on the board and the SmartRF Studio takes direct control of the hardware through the JTAG to set the device to listen for incoming Zigbee packets.

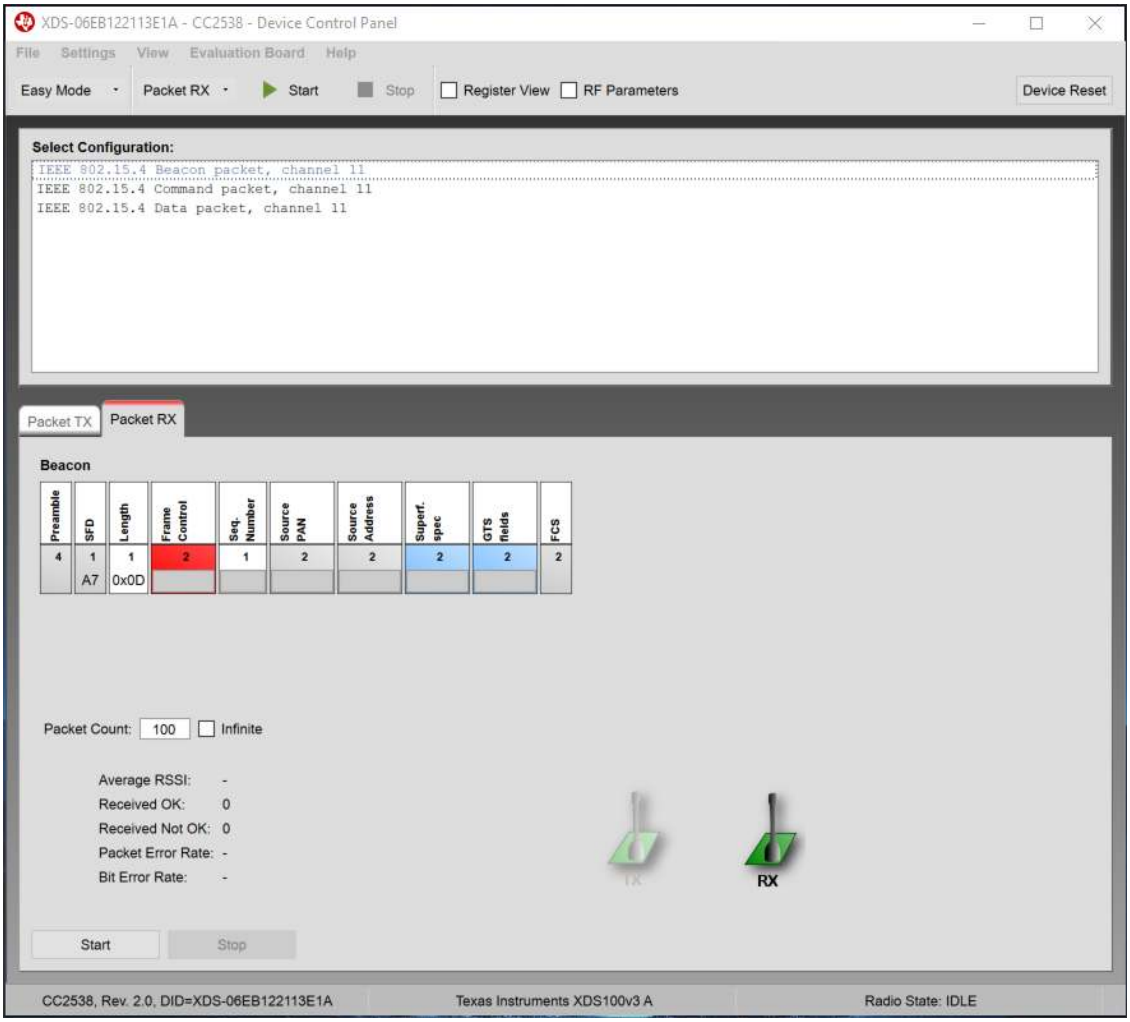


Figure 10: Setting the SmartRF06 board to receive packets.

For the device under test, the JTAG programmer was connected to another host computer and the SmartRF Studio was started on that machine. This device was set to *Packet TX* mode and started.

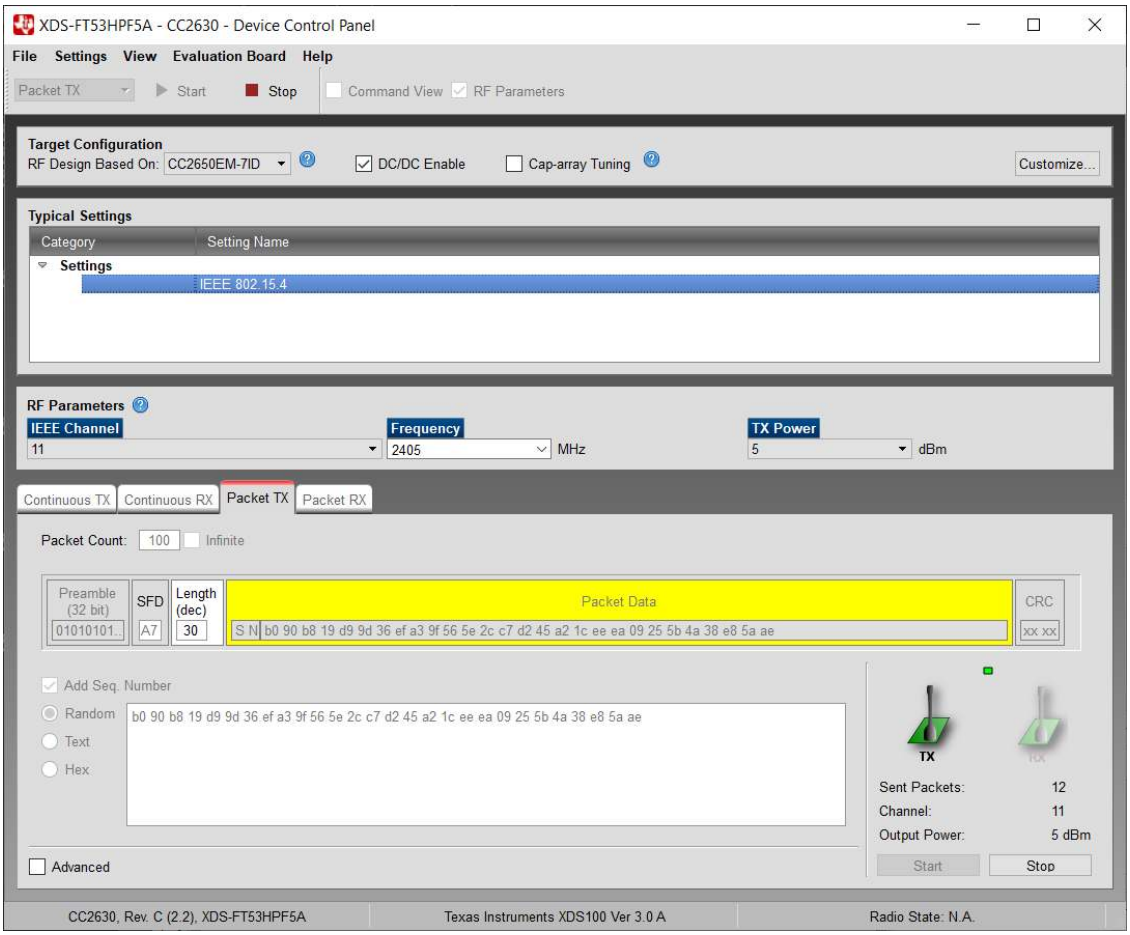


Figure 11: Setting the NeTs Mote to transmit packets.

3.5 Results

For the LED test, the NeTs Mote was programmed using the Contiki NG operating system, including board firmware. The program successfully controlled the IO pins to turn the LED on and off.

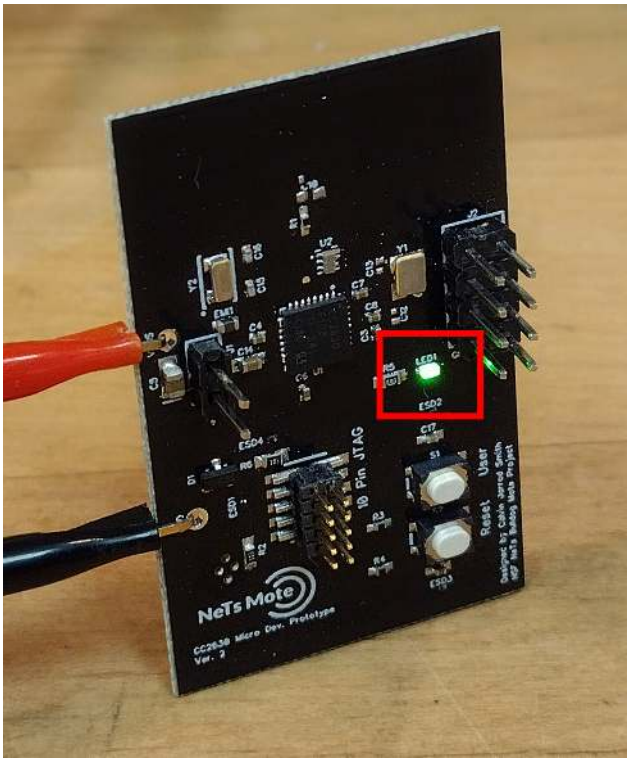


Figure 12: Enabling the on-board LED.

During the testing for the UART, there was no communication that could be established between the NeTs Mote and the host computer. So this error will need further investigation and troubleshooting.

However, for the antenna test, the SmartRF06 module received all 100 transmitted test packets without error with an average RSSI of -41.8 dBm. This shows that the antenna circuit and balun work correctly and no further testing is needed.

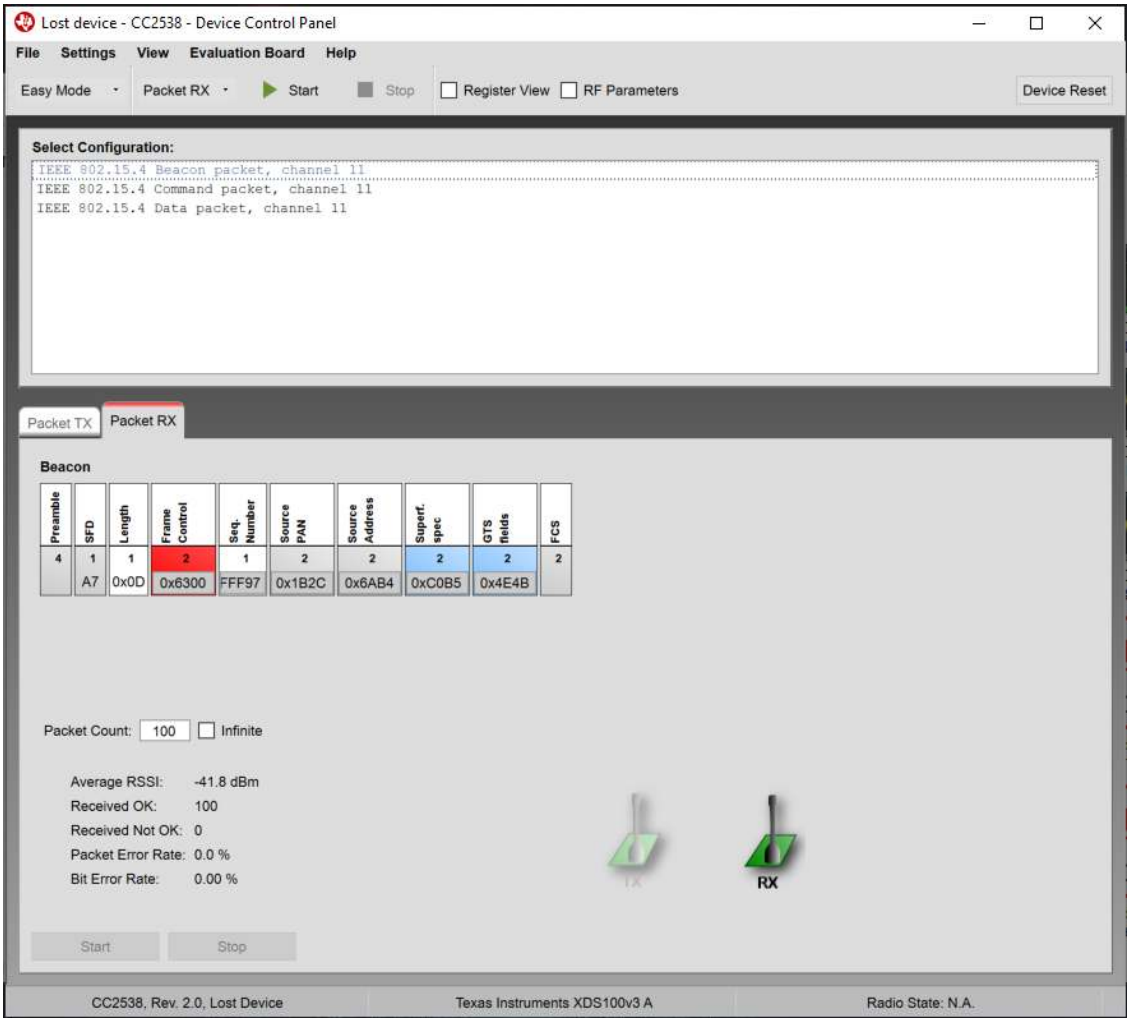


Figure 13: Successful reception of NeTs Mote's 100 packets.

The NeTs Mote has been improved and is making progress toward a fully functioning sensor mote with custom PCB design and PCB antenna. Further iterations will be necessary for this development.

4 SM14 Bulldog Mote Shield

4.1 Overview

While the antenna circuit and completely custom system of the NeTs Mote was being designed and tested, another device was designed that focused on a system's power supply and sensors. As a precursor to designing a working sensor mote, a development board was chosen that contained a widely available wireless microcontroller module. This abstracted antenna design and testing which permitted this project to focus on the other aspects of wireless sensor mote design.

4.2 Board and Wireless Module Selection

To keep the focus of this part of the project on power and sensors, the SM14Z2538PA1V3.0 board was selected for its use of the CC2538 module. This module contained a TI CC2538 chip, which is another low-power wireless microcontroller that uses Zigbee or 6LoWPAN protocol. This was important because this system needed to be compatible with the TelosB or Sky Motes that were previously purchased for the project. As well, the CC2538 module contained all of the microcontroller components needed to be operational, such as system clocks, passive components, and antenna circuitry and balun. This module was also designed with an meandering inverted-F antenna which allowed a smaller form factor than the traditional inverted-F PCB antenna, despite having a less ideal radiation pattern.

In addition, this module was designed with the CC2592 Range Extender, which helps amplify input and output 2.4 GHz radio transmissions. This range extender can be set to high-gain mode or low-gain mode for the receiver and amplifies transmission power depending on the received power output from the microcontroller.

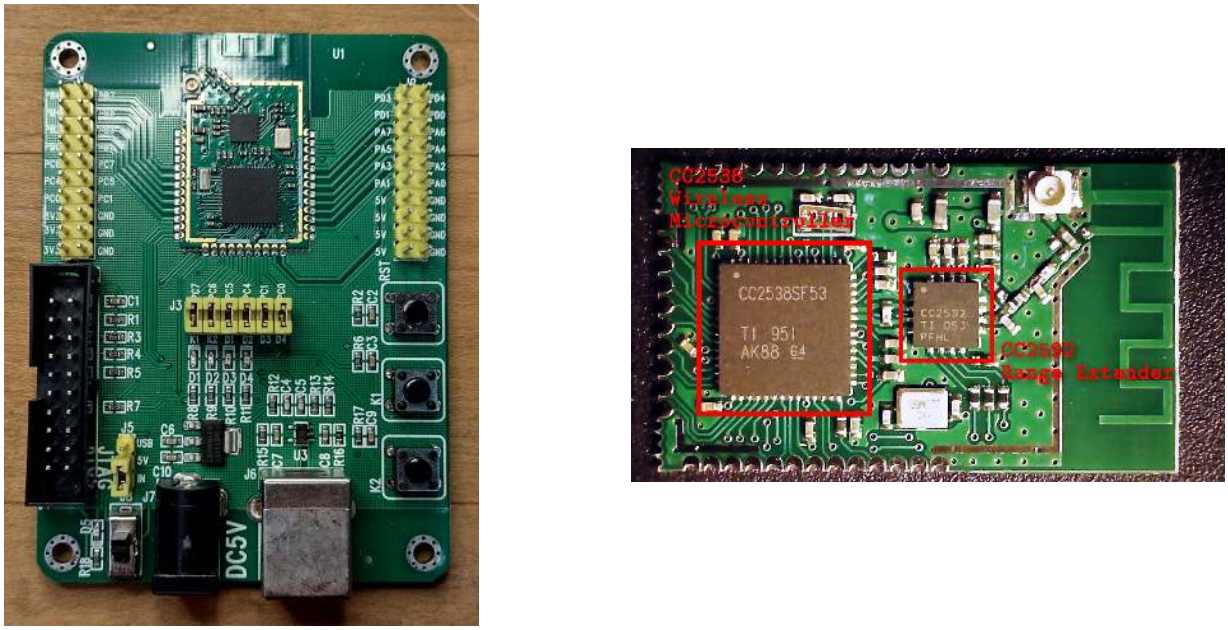


Figure 14: SM14 Development Board and CC2538 Module.

This development board contained header pins that made it possible to devise a *shield*, which is a PCB system that can sit on top of the board, fitting on those header pins, and allow additional features to the board.

4.3 Sensor Selection

The use of sensors on the shield was the first concern since this was to be made into a sensor mote. Three environmental phenomena were chosen as parameters for sensing: air quality, ambient light and temperature. Several sensors were researched and the following sensors were chosen: MQ135 for air quality sensing, BPW34 for ambient light sensing, and BMP280 for temperature sensing. The MQ135 was chosen for its ability to sense a wide range of chemical contaminates such as benzene, ammonia, carbon monoxide, alcohol, or smoke. The BPW34 was chosen because firmware was already written in Contiki NG operating system for it. As well the BMP280 was chosen because of its reliability and low cost.

Figure 15: The MQ135, BPW34 and BMP280 Sensor Devices.

4.4 Device Power

The device power needed to be considered after sensor selection. Two of the three sensors, the MQ135 and BPW34, were analog sensors that required 5 V power. The microcontroller and the BMP280 required 3.3 V power to operate. These two different voltages required different voltage regulation. It was decided to use a higher voltage battery, then use switching regulators (or buck converters) to efficiently step-down the voltage from the battery to make the system more efficient.

TI's Webench® Power Designer was used to design the two voltage regulators. The designs were chosen to be 5 V and 3.3 V with a max of 1 A for each. Out of the various options and designs, two were settled upon based on chip availability on the electronics supplier, Mouser's website and overall fewest Bill of Materials (BOM) count. The TLV62150 and TPS561201 buck converter chip were chosen for 3.3 V and 5 V respectively. The power supply circuits given by TI's Webench are shown below. To power these supplies, a 9 V battery cell was decided upon because of its relatively compact size and having a voltage greater than the highest needed for the system.

To determine the efficiency of each buck converter, the current consumption of each device needed to be determined. The current draw of each device was tabulated from either their respective datasheet or from measuring the device's current draw at steady-state and is shown in Tables 1 and 2:

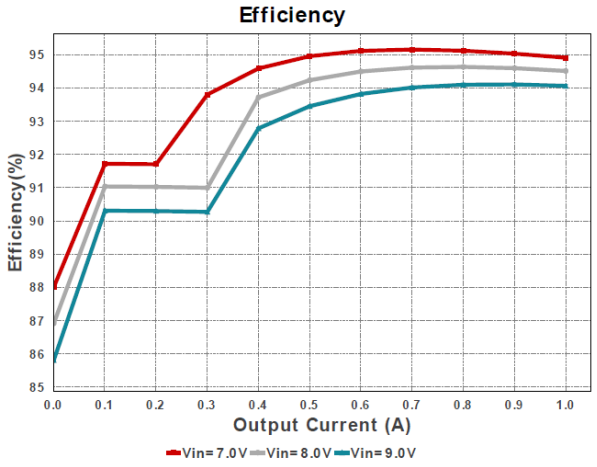
CC2538 Components	**Consumption (mA)**
Core	13.00
Radio (TX)	34.00
General Purpose Timer	0.12
I2C	0.10
UART	0.70
ADC	1.20
Total current consumption	49.12

Table 1: Current Consumption of each CC2538 Component based on the CC2538 Datasheet [4].

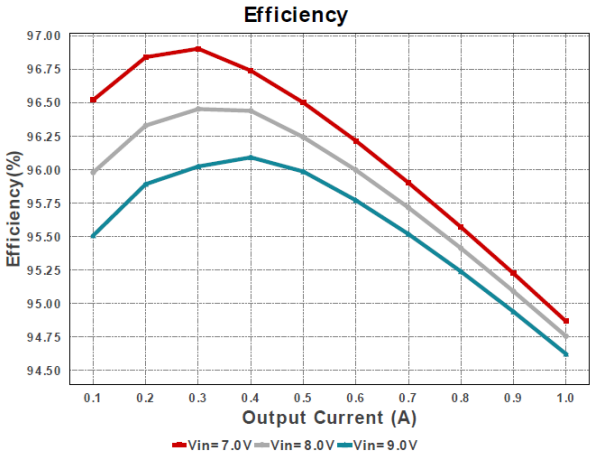
Device	Consumption (mA)	Device Voltage (V)
CC2538 Compenents	49.12	3.3
CC2592	155.00 (max)	3.3
BMP280	1.12	3.3
MQ135	110.00	5.0
ALS	0.10	5.0
LEDSx4	8.00	3.3
Total 3.3 V current consumption		**213.24**
Total 5 V current consumption		**110.10**

Table 2: Current Consumption of all board devices.

Using the charts given for each respective switching regulator the efficiency of each device can be determined when at maximum current usage:



(a) TLV62150RGTR Efficiency for 3.3 V Devices [5].



(b) TPS561201DDCR Efficiency for 5 V Devices [6].

Figure 16: Efficiency Charts for each Switching Voltage Regulator.

These chart indicate the efficiency at maximum current consumption with a fully charged 9 V battery is approximately 90.2% for the 3.3 V converter and 95.55% for the 5 V converter.

4.5 Circuit and PCB Designs

The TI Webench® Power Designer recommended circuit designs for the TLV62150 IC (3.3 V buck converter) and the TPS561201 (5 V buck converter) are shown in Figures 17 and 18:

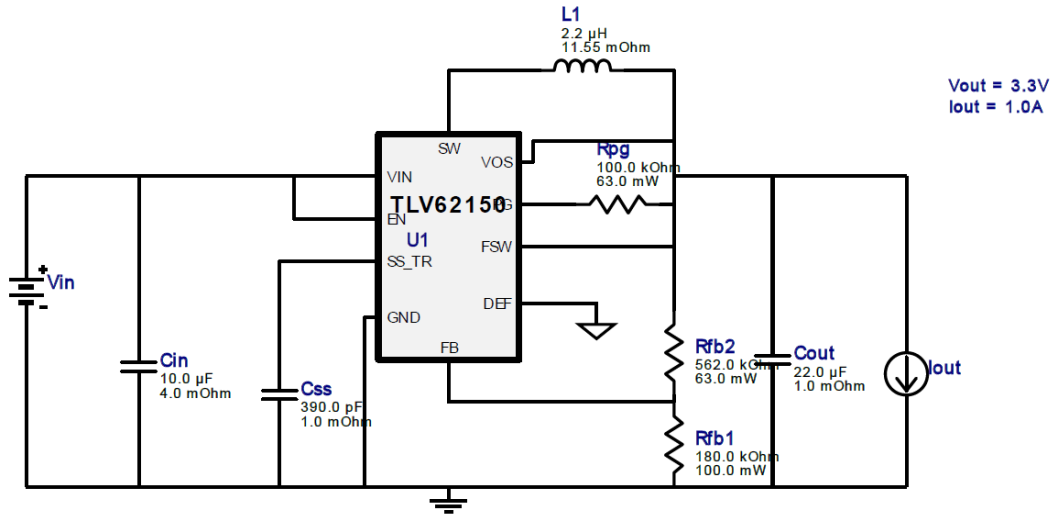


Figure 17: Recommended Circuit Layout for the TLV62150 IC for 3.3 V Output.

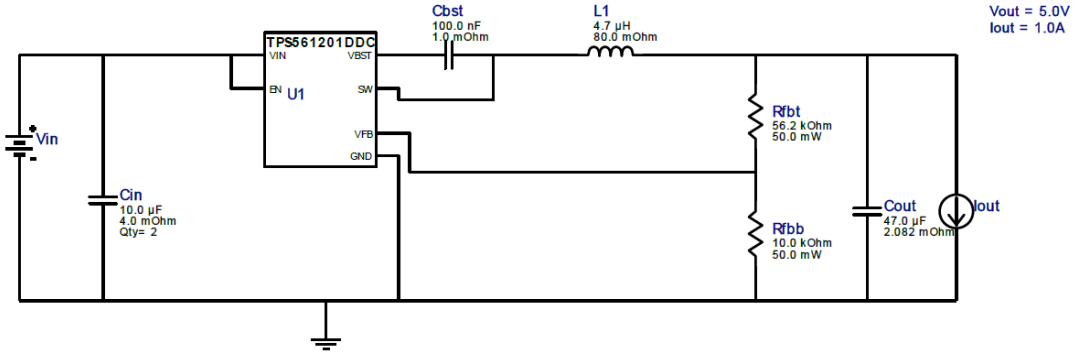


Figure 18: Recommended Circuit Layout for the TPS561201 IC for 5 V Output.

Given the circuit schematic for the power supplies, the circuit design of the rest of the system was completed using the recommended layouts from each device's manufacturer. The completed PCB design for the SM14 Sheild is shown in Figure 38:

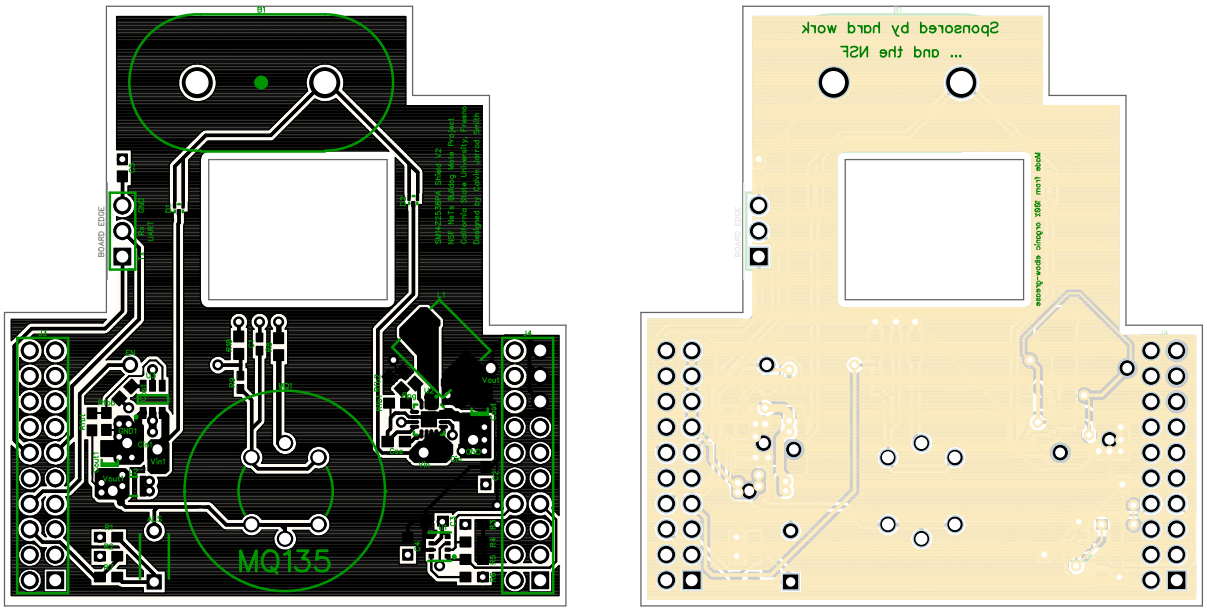


Figure 19: PCB Layout of SM14 Shield.

Separate input diodes were used to direct current into the two voltage regulators. To determine the maximum current draw through the input diode, the current consumption at the output and the efficiency of each regulator needs to be known at their maximum draw. These values were found in the previous section. They can be applied to the following equation:

$$P_{out} = P_{in} \times \text{Efficiency} \tag{1}$$

The input and output power can be calculated using the following diagram:

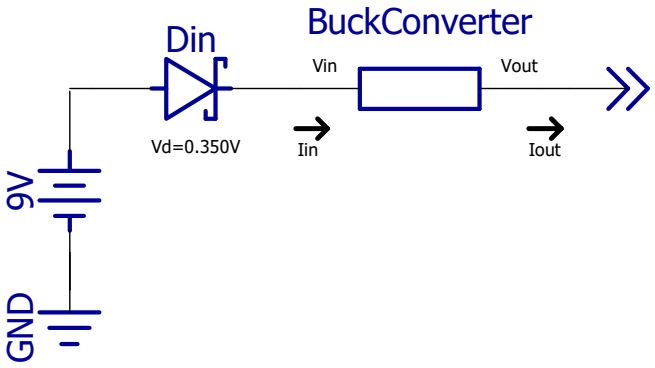


Figure 20: Basic Schematic of Buck Converter.

To find I_{in}, the formula for power:

$$P = V \times I \tag{2}$$

in equation (1):

$$I_{in} = \frac{P_{in}}{V_{in}} = \frac{P_{out}}{V_{in} \times \text{Efficiency}} = \frac{V_{out} \times I_{out}}{V_{in} \times \text{Efficiency}} \tag{3}$$

With this equation the input current for the 3.3 V regulator and 5 V regulator were 90.19 mA and 66.6 mA, respectively, during the maximum current draw. Therefore the Toshiba 1SS385FVL3F Schottky Diode was chosen as the input to each regulator as its maximum forward current was 100 mA.

As well, voltage dividers were used on the outputs of the MQ135 and ALS to drop the analog voltage range from 0-5 V to 0-3.3 V.

4.6 Firmware Development

Firmware for the CC2538 chip already existed in the Contiki NG operating system. However, the firmware for the BMP280 and the CC2592 needed to be written, as well as ADC pins to read the ambient light sensor (ALS) and the MQ135, and an extra IO pin to control the 5 V switching regulator.

The CC2592 requires the control of three pins to determine when transmitting and receiving, and whether the receiver is in high-gain mode or low-gain mode. The pin inputs and controls are shown in table 3.

PA_EN	**PA_EN**	**HGM**	**Mode of Operation**
0	0	X	Power down
X	1	0	RX Low-gain Mode
X	1	1	RX High-gain Mode
1	0	X	TX

Table 3: CC2592 Control Pins [7].

The control sequences for the `PA_EN`, `LNA_EN` and `HGM` had to be placed in the correct locations to change the modes before and after transmissions, and when the system first starts. First, under `arch/platform` the directory for the `cc2538dk` was copied and name `bulldogV1` for the new mote platform being made. Then within this new folder, the `platform.c` file was modified. This file, `platform.c`, handles all of the system initialization and sleeping sequences. The pins to control the CC2952 chip were initialized in the `platform_init_stage_three()`, which is the last initialization function of the system. All three pins were configured as outputs, then the `PA_EN` pins was set to 0 and `LNA_EN` was set to 1 to set the system to begin in RX mode. The high/low-gain mode was set by using a conditional directive that checks if the constant `CC2538_RF_RE_HGM_EN` is 1 or 0 and sets the `HGM` pin accordingly.

For changing the pins depending on the mode of operation, the chip firmware needed to be modified. All microcontroller (as opposed to platform) firmware is located in {{`Contiki-`

Home}/arch/cpu. For the CC2538 chip, the file cc2538/dev/cc2538-rf.c was adjusted in the transmit() function for this purpose. Since this file dealt with the low-level chip drivers the device registers were handled manually to set and reset the LNA and PA pins. The TI CC2538 Datasheet was used to determine the algorithm needed to correctly set and reset these pins and was implemented using Contiki NG's REG() function call to write to the chips registers. The PA pin was set to 1 and LNA to 0 during the transmission, and reset (PA to 0 and LNA to 1) after the transmission was complete. This would handle switching from RX to TX mode, and back again.

The BMP280 firmware was written and placed in the {Contiki-Home}/arch/platform/-bulldogV1 directory along with the other sensor firmware. The format for calling the driver functions was copied identically to the existing drivers to make using existing Contiki NG sensor data acquisition seamless. This platform however, did not contain an I2C protocol driver to communicate with the BMP280, so these drivers were also written using the exact hierarchy as in other platforms in Contiki NG that already had I2C drivers written. The remainder of the BMP280 drivers were written using Bosch's BMP280 datasheet algorithms and recommendations [8].

In addition, the analog sensors themselves needed to be turned off to save energy. To do this, the *enable* pin of the 5 V buck converter was used to turn the ALS and MQ135 off. This pin was wired to the CC2538 module on pin A4. The MQ135 and ALS sensors had their own firmware written using other analog sensor firmware in Contiki NG. Within each devices firmware in mq135-sensor.c and als-sensor.c, the configure function was altered to more effectively turn and off the voltage regulator. The configure() function would help in this case and has many purposes and is called on several occasions. It has two parameters: *type* and *value*. When the system is first started up and the sensors need to be configured for the first time before use, the configure() function is called on each sensor and the *type* parameter is passed with a value SENSORS_HW_INIT. This is useful to set hardware pin values, and write to hardware registers if using a digital sensor. The other event that requires the configure() function to be called is when a sensor is when a sensor is activated or deactivated in the Contiki application file using SENSOR_ACTIVATE(...) or SENSORS_DEACTIVATE(...) respectively. When these functions are called the *type* passed to configure() is equal to SENSOR_ACTIVATE. Then *value* is used to determine if the sensor is activated or deactivated. For both the ALS and MQ135 these concepts were used to write the firmware to enable and disable the 5 V regulator.

It was observed during initial testing that obtaining a single sensor sample (from the ALS and MQ135) resulted in very inconsistent readings. The firmware was modified to obtain many samples equal to a power of 2 (128), and use a right shifting operation to average out the readings. This produced much more consistent and smoother changes in the sensor data. It was later determined that the cause of this was likely a large voltage ripple due the switching action of the 5 V buck converter. The ripple was significantly reduced in the Bulldog Mote design.

Lastly, the UART pins of the board needed to be changed to be connected to UART

header pins on the SM14 shield. The header pins for UART RX and TX were connected to pins A0 and A1 on the CC2538 module, respectively. The pins were changed in Contiki under {`Contiki-Home`}`/arch/platform/dev/board.h`. This file contains all of the platform's pin names and was used to rename the LED pins, MQ135 and ALS pins, I2C pins and CC2592 pins.

All firmware for the SM14 Shield and Bulldog Mote is included in Appendix A.

4.7 Testing Procedure

To test the functionality of the SM14 Shield, three separate tests were performed: a voltage regulator test, a sensor test using the UART to read the sensor data, and a wireless networking test to transmit a specific sensor's data through a network to a server. The voltage regulators needed to be tested to ensure corrected voltage level for the separate systems. Also, each sensor needed to be tested to see if it was operating sufficiently, to do this a desk light was used hovering over the board to test the ALS, a bottle of rubbing alcohol was used to test the MQ135, and touching the metal of the case to warm up the BMP280 temperature sensor. Rudimentary sensor tests as this were the only option as a temperature controlled or light controlled environment was not possible. Neither was it possible to access to a chemical hood with precise control in parts-per-million of air contaminates such as alcohol, carbon monoxide, benzene, or ammonia.

Test points were placed on the board to test the power systems. To start, the SM14 shield was not connected to the header pins of the SM14Z2538PA1 board. The shield's 9 V battery connectors were connected to an external DC power supply set to 9 V, and a multimeter was used to read the *Vout* test point of the 3.3 V voltage regulator.

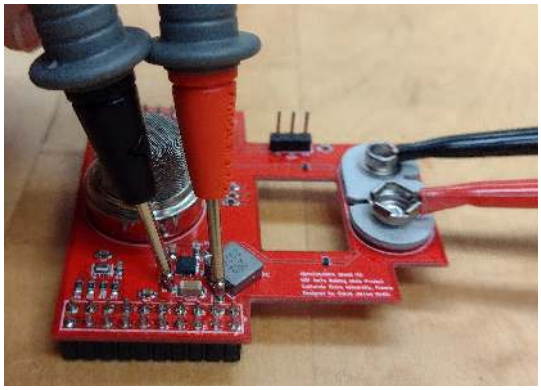


Figure 21: Voltage Reading of the 3.3 V Step-Down Converter.

Next, the 5 V regulator needed to be tested. To do this, the converters *enable* needed to be connected to the 3.3 V regulator output to turn the system on and ground to turn the system off. This was done and the multimeter was connected to the 5 V buck converter output.

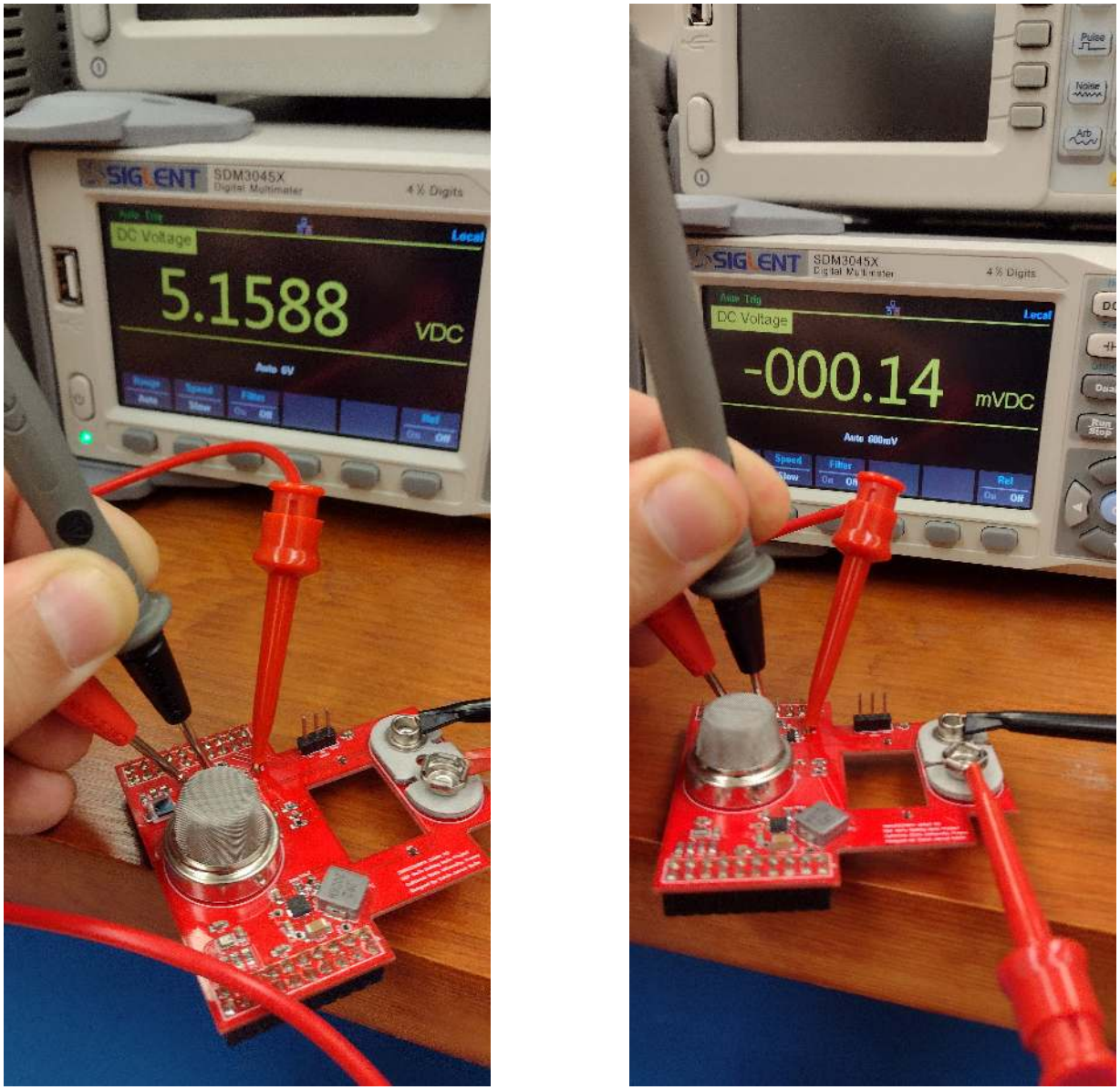


Figure 22: Voltage Reading of the 5 V Step-Down Converter with enable pin high (a) and enable pin low (b).

To test the sensors, a program was written using Contiki NG to read all of the sensors and output the data over UART. Next, shield was fitted on the top of the SM14Z2538PA1 pins, then the combined system was connected to a host computer through Olimex TMS320-XDS100v3 JTAG programmer, and the UART pins connected to the USB-to-TTL adapter just like the NeTs Mote during testing.

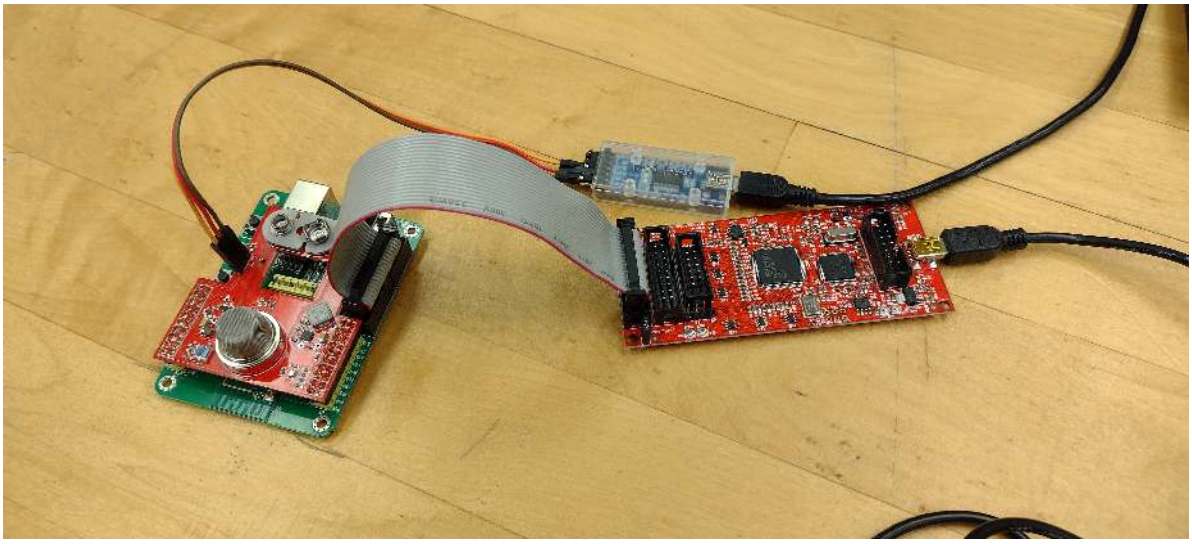


Figure 23: Combined SM14 System Connected to host computer via JTAG and USB-to-TTL Adapter.

For the sensor test, a desktop light was used to change the readings of the ALS. The UART messages were read using RealTerm software.

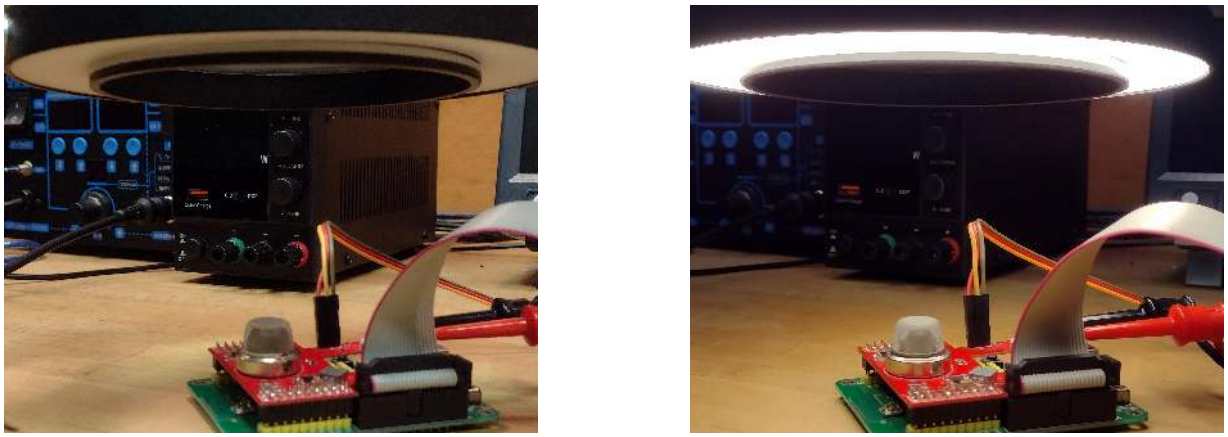


Figure 24: Using desk light to increase the luminous flux.

Next, the MQ135 was tested. After the system had warmed up and the MQ135 readings had stabilized, the sensor was introduced to vapors from isopropyl alcohol.

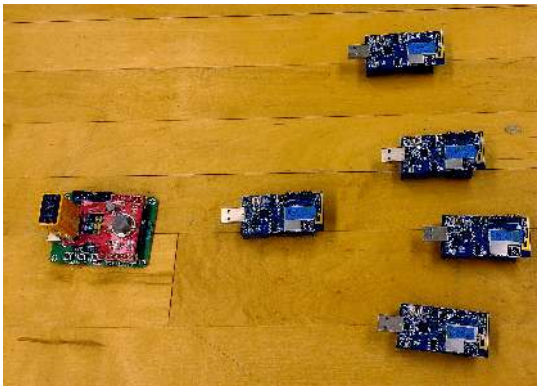
Figure 25: Using Isopropyl Alcohol as sensing phenomena for MQ135. Alcohol bottle cap filled with alcohol and placed next to sensor (b).

Then the temperature sensor was tested by allowing it to run at room temperature for about 20 minutes, then touching the sensor's metal casing to increase the temperature readings.

Figure 26: Touching the BMP280 Sensor to increase measured temperature.

The last test was to place the SM14 system into an existing wireless sensor network and successfully transmit data to a data server. A program in Contiki NG was written for a *UDP client* to transmit temperature data to a *UDP server*. Several TelosB motes were programmed in the same fashion and a one was also used as the server itself. The server was connected directly to the host computer to read the output messages from the networked motes.

(a) TelosB Mote used as Server.



(b) TelosB Client Setup with SM14 System forming a network.

Figure 27: TelosB Network Setup.

4.8 Results

Figure 28 shows the completed SM14 Shield System on top of the SM14Z2538PA1 development board.

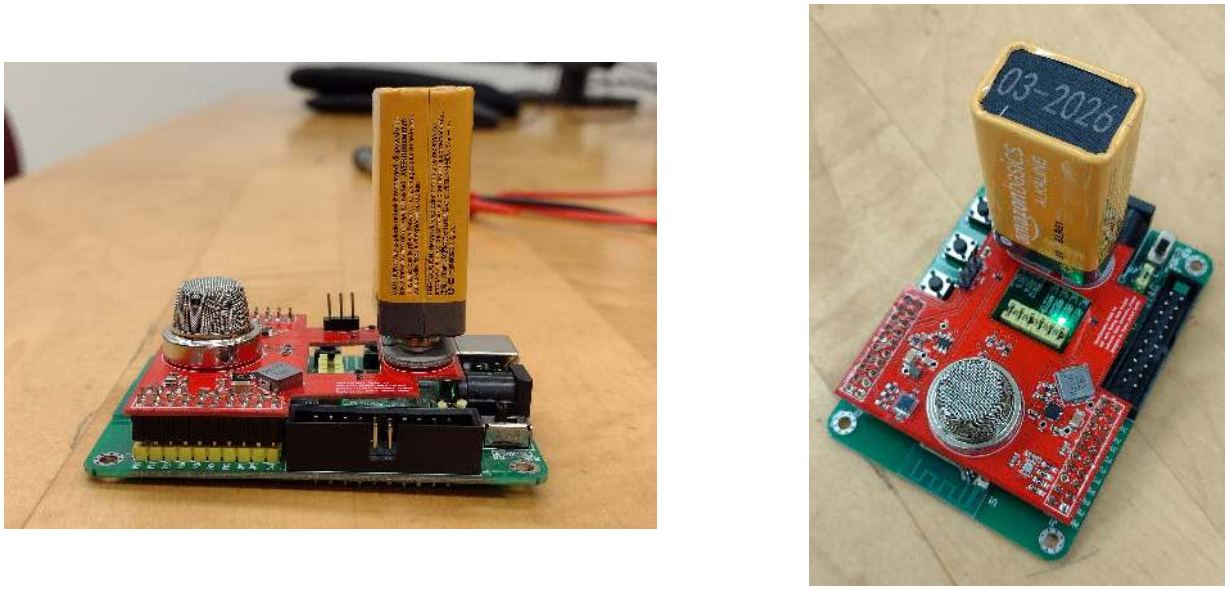


Figure 28: The SM14 Shield System.

As seen in figures 21 and 22, the power systems measured within 10% of expected voltages and do not exceed the manufacture's maximum voltage ratings for either system.

As seen in figures 29, 30 and 31, the ALS and MQ135 do detect fluctuations in light and alcohol levels, and the BMP280 does correctly measure temperature. However, the MQ135 and ALS are not *calibrated*. Even though these sensors are giving readings that vary with changing phenomenon, it isn't an accurate method to measure luminous intensity (lux) or air-quality indication (AQI). For this, accurate measurement devices and controlled environments would be required to test and tune these sensors by selecting the correct circuit resistances and/or software calibration.

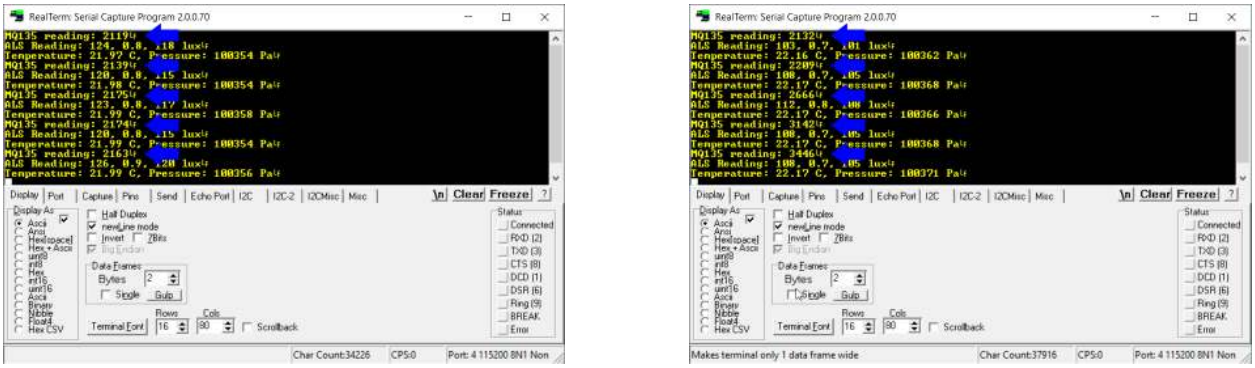


Figure 29: UART Readings for the MQ135 Sensor while being exposed to Isopropyl Alcohol.

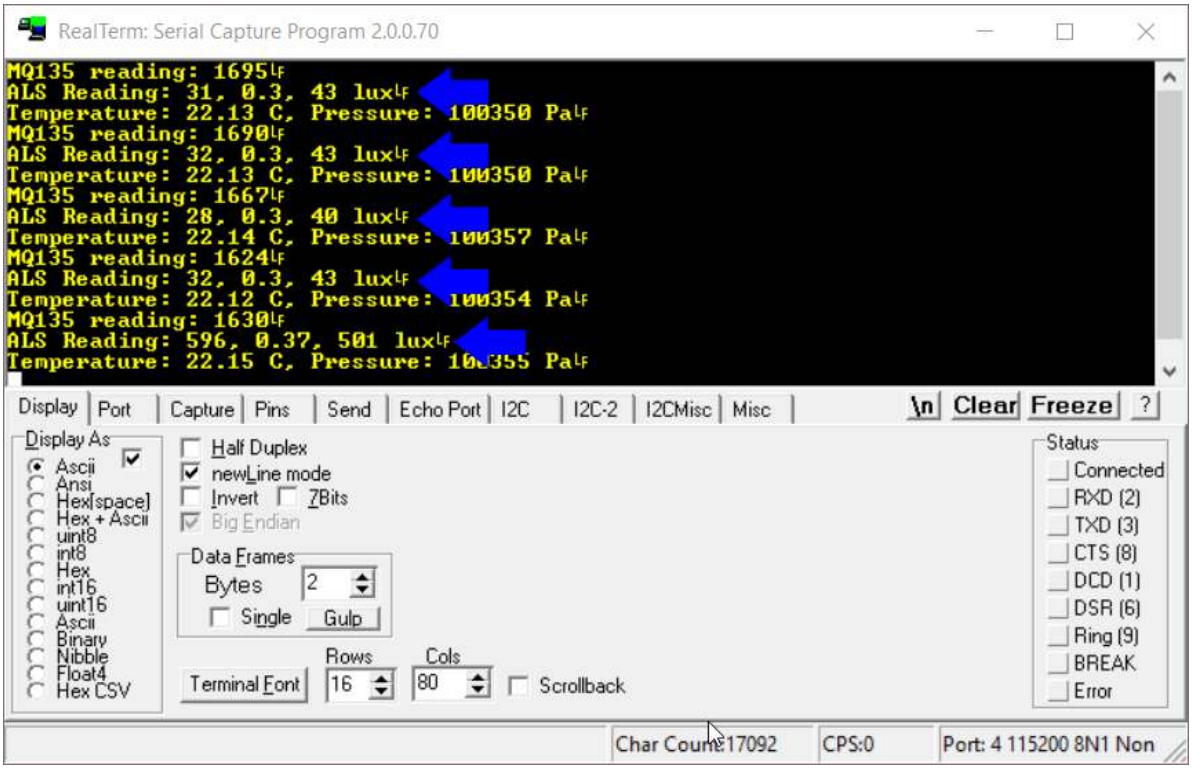


Figure 30: UART Readings for the ALS while being exposed to the desk lamp.

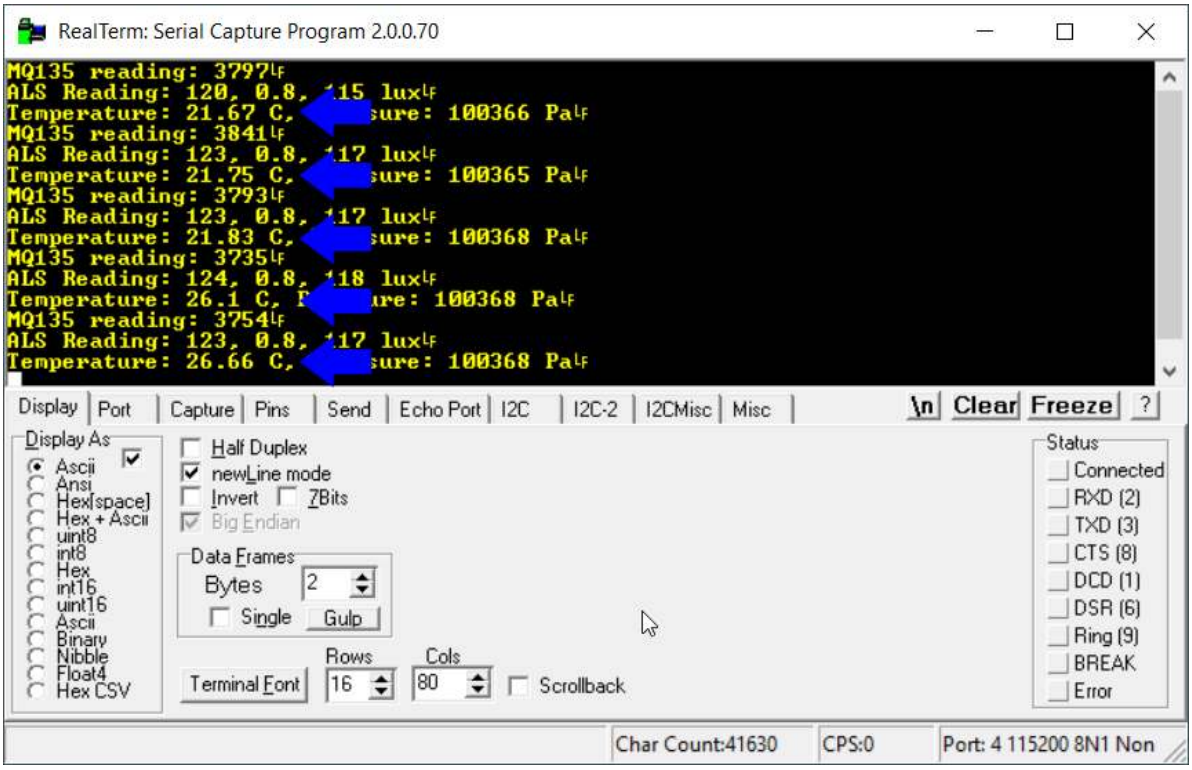


Figure 31: UART Readings for the BMP280 while metal case was being touched.

As well, figure 32 shows the temperature data from the SM14 System was obtained and displayed. Successful reception of the SM14 system's readings demonstrates the correct operation of the CC2592 range-extender chip. Despite the SM14 system not seemingly *receiving* any data, packet reception on the SM14 system is necessary for any data transmissions to be received because of the formation of the RPL DODAG used for network routing [9].

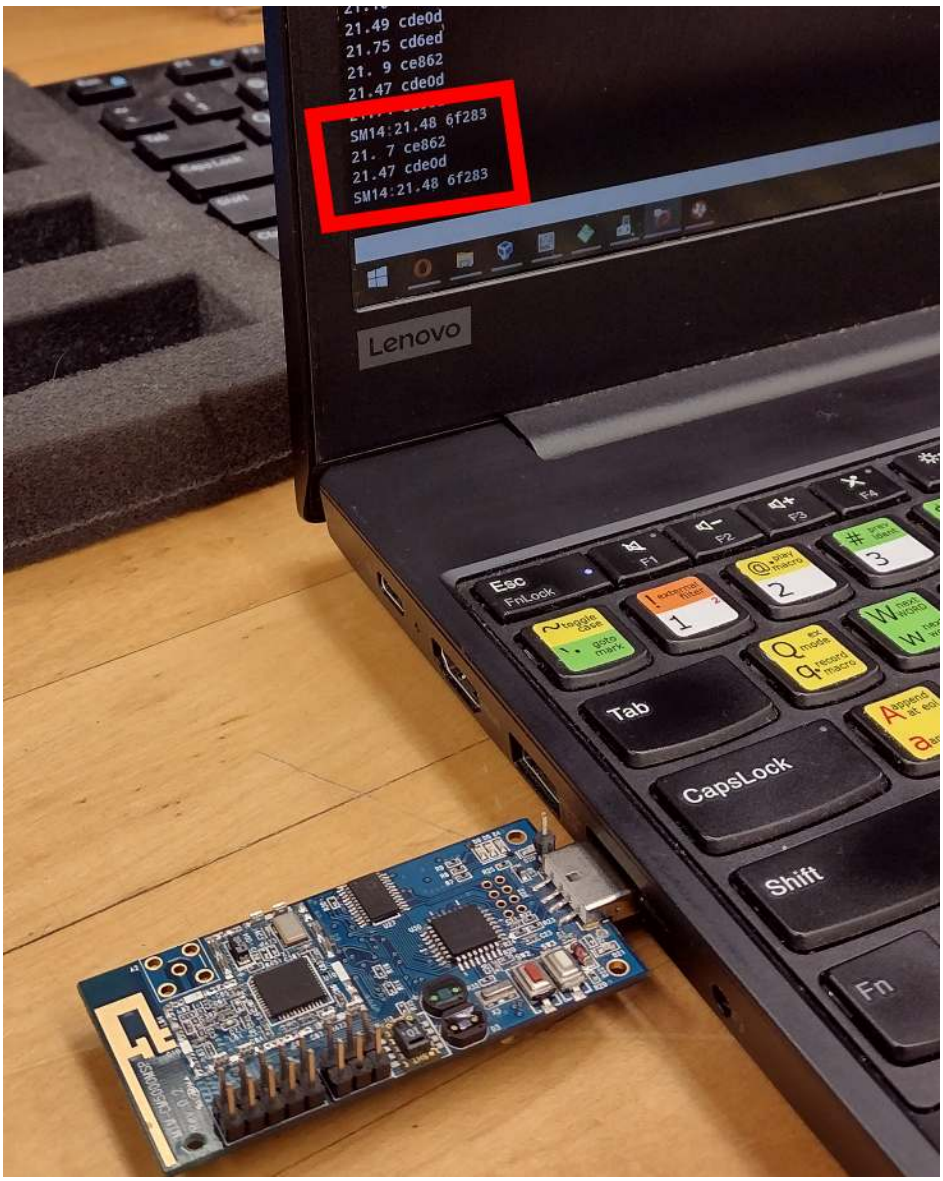


Figure 32: TelosB Server reading temperature data from network and identifying SM14 System.

Overall, SM14 system worked well as a first-step to creating the Bulldog Mote. Instead of focusing on the microcontroller and board components, the SM14Z2538PA1 platform was used as a base of the SM14 Shield. This allowed the focus to be placed on sensor selection and power.

5 Bulldog Mote

5.1 Overview

The SM14 Shield was a stepping stone to creating a new mote system. The Bulldog Mote is the extension that takes the SM14 Shield into a custom, mobile mote system. Modifications were made to the power system to correct potential issues with the switching regulator output and external memory was added. The sensors and other peripherals were connected to the CC2538 module through the same pins as the SM14 board to keep the firmware the same for both boards.

5.2 Low-Dropout Regulators and Ferrite Bead Filters

Switching regulators aim to generate a stable DC voltage which can be selected by the use of passive components. The act of switching in these regulators creates voltage ripples at the output with a frequency equal to that of the regulator switching frequency. As well, the output of these systems can contain voltage spikes that are created from the duration of the transition times during the switching. In sensitive systems, such as an RF transceivers, this voltage ripple and spikes need to be reduced as much as possible. Output capacitors do filter out some of this noise, however, some additional filtering might be necessary [10].

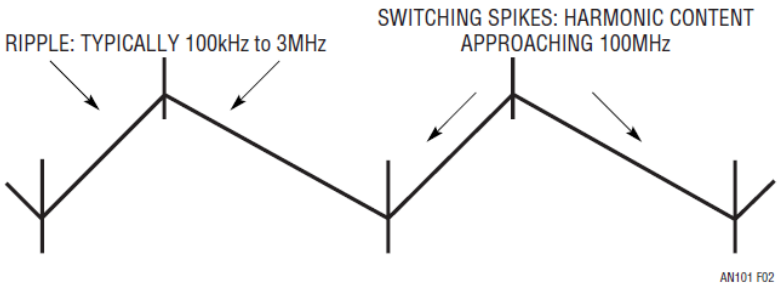


Figure 33: Voltage ripple caused from switching regulator [10].

One solution is to use a Low-Dropout (LDO) or Linear Regulator with input and output capacitors, along with ferrite bead filter. The LDO layout that is recommended by the Linear Technology application note, *Minimizing Switching Regulator Residue in Linear Regulator Outputs* is shown in figure 34

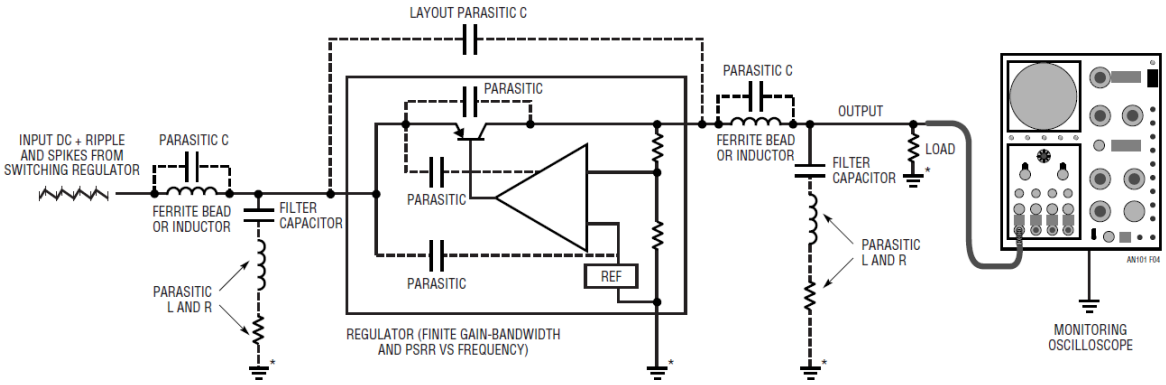


Figure 34: Circuit Layout of LDO Filter Circuit showing Parasitics [10].

This diagram shows not only the recommended circuit layout, but also the component parasitics that may need to be considered during the design phase. The moral of the story is that despite how well the circuit is designed to reject the ripple and noise, it won't be perfect and there will always be a small amount of ripple that makes it through the circuit because of parasitics. However, applying an LDO plus ferrite bead filter will greatly attenuate whatever voltage ripple and spikes do exist.

To remedy some parasitics, a combination of tantalum and ceramic capacitors were used in the Bulldog Mote design. Ceramic capacitors have the benefit of having extremely low effective series resistance (ESR) and effective series inductance (ESL), as well as a decent capacitance per unit volume and very low cost. Tantalum capacitors have much better resiliency to vibration and better voltage stability compared to ceramic capacitors but do not benefit from the same low ESR and ESL, and are much more expensive [11]. So for the Bulldog Mote design, tantalum capacitors were chosen as the output capacitors for both voltage regulator circiuts (3.3 V and 5 V) for the benefit of voltage stability and the remainder of the capacitors were left as ceramic to keep costs low.

The specific LDOs were chosen based on the maximum current needed from each regulator (table 2) and to minimize voltage drop-out a much as possible to increase efficiency as LDO efficiency is rated as:

$$\text{Efficiency} = \frac{I_{out}}{(I_{out} + I_{GND})} \times \frac{V_{out}}{V_{in}} \times 100\% \tag{4}$$

Where I_{GND} is the quiescent current of the LDO regulator.

The TPS7A0530 was chosen for the 3 V system, lowering system voltage to 3 V instead of 3.3 V to keep the 3.3 V switching regulator components the same, and TPS7A2050 for the 5 V system. However the output of the 5 V switching regulator was changed to 5.29 V to maintain the 5 V system voltage. The equation from the TPS561201 switching regulator's

datasheet was used to calculate the new resistor values:

$$V_{out} = 0.768 \times \left(1 + \frac{R1}{R2}\right) \tag{5}$$

Keeping $R2 = 10k\Omega$, and using 5.3 V output as an approximation, equation (5) can be rearranged to find $R1$:

$$R1 = R2 \times \left(\frac{V_{out}}{0.768} - 1\right) = 10000 \times (5.3/0.768 - 1) = 59010.42\ \Omega \approx 59\ k\Omega \tag{6}$$

Then reapplying equation (5) to find the precise output voltage of the switching regulator:

$$V_{out} = 0.768 \times \left(1 + \frac{59\ k\Omega}{10\ k\Omega}\right) = 5.2992\ V \approx 5.3\ V \tag{7}$$

Increasing the voltage output of the TSP561201 was necessary since the TSP7A2050 LDO regulator placed at its output requires a minimum *dropout* voltage, which is a minimum voltage difference between the input and output voltage of the LDO regulator needed to maintain its nominal voltage output rating [12].

Given the input and output voltages to each LDO, switching regulator efficiencies, system current demand and LDO quiescent current from the manufacturers datasheets, the overall efficiency of each power system (3 V and 5 V) can be determined. To start the efficiency of the combined switching regulator and LDO power supply can be determined as such:

$$E_{supply} = E_{LDO} \times E_{switching} \tag{8}$$

Plugging equation (4) into (8), the 3 V system efficiency can be determined using the quiescent current from the TPS7A0530 [13]:

$$E_{Supply\ 3V} = \frac{0.21324}{0.21324 + 6 \times 10^{-6}} \times \frac{3}{3.3} \times 0.902 \times 100\% = 82.0\% \tag{9}$$

and the efficiency for the 5 V power system can be determined once the quiescent current is found. The chart giving the quiescent or ground current of the LDO can be found in the TPS7A20 datasheet:

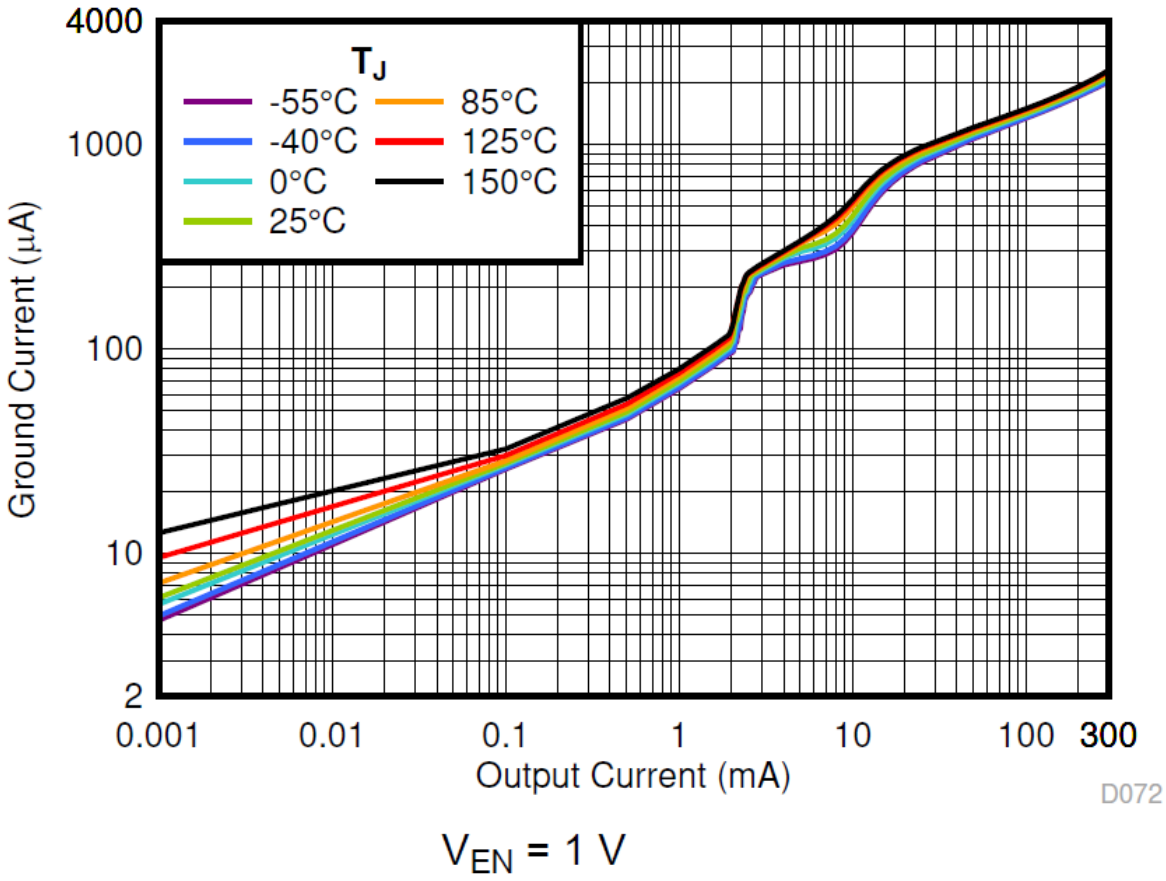


Figure 35: TPS7A20 Ground Current Versus Output Current [14].

With the output current being 110.10 mA for the 5 V power system (table 2) the ground current should be approximately 1.6 mA at room temperature:

$$E_{Supply\ 5V} = \frac{0.1101}{0.1101 + 1.6 \times 10^{-3}} \times \frac{5}{5.3} \times 0.9555 \times 100\% = 88.9\% \tag{10}$$

Comparing these to the previous design, both power systems are less efficient that in the SM14 Shield design. Despite this, the power systems will have less problems with voltage ripple that can cause disturbances with the RF circuitry.

5.3 External Flash Memory

To help increase battery life, the system should be placed in a very low-power state for as long as possible. In most microcontrollers, such as the CC2538, the lowest sleep state also erases any data and variables in RAM and resets the programmed application. Unfortunately this also erases the routing information of the network created by the Contiki NG software, which would force a rediscovery of a network path back to the sink device on every wake-up. To remedy this, an external flash memory module was included on the Bulldog Mote system. The Adesto AT25SF321B NOR flash memory module was chosen for its low cost, large size (32-Mbits or 4 Mbytes), and low-power standby current (25 μA) [15].

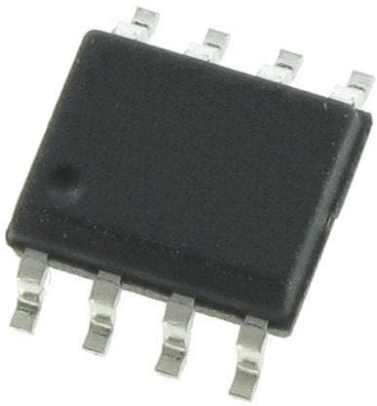


Figure 36: The AT25SF321B NOR Flash IC by Adesto.

The device uses SPI with single, dual or quad I/O support for even faster data transmission. SPI firmware and flash memory firmware had to be written as well to operate the memory unit in Contiki NG.

5.4 Circuit and PCB Designs

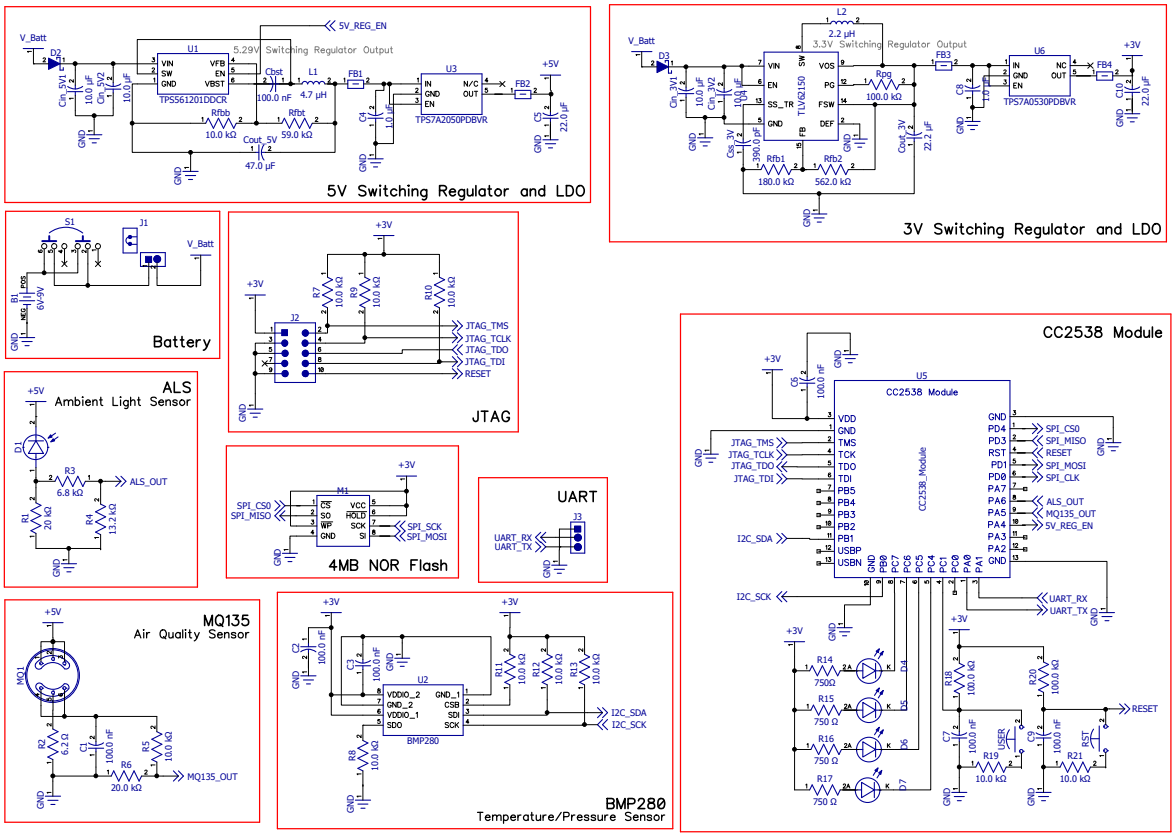


Figure 37: Circuit Layout of Bulldog Mote System.

A couple changes were made to the Bulldog Mote from other previous designs. First, the JTAG pins were changed from the standard 20-pin, 2.54 mm header pins, to the 10-pin ARM JTAG using 1.27 mm header pins. This greatly reduces the interface size, despite needing an additional adapter for the Olimex TMS320-XDS100v3 JTAG programmer. Also, a power switch was included in this design, unlike the TelosB motes, which require manually removing a AA battery from the battery pack to turn the device off.

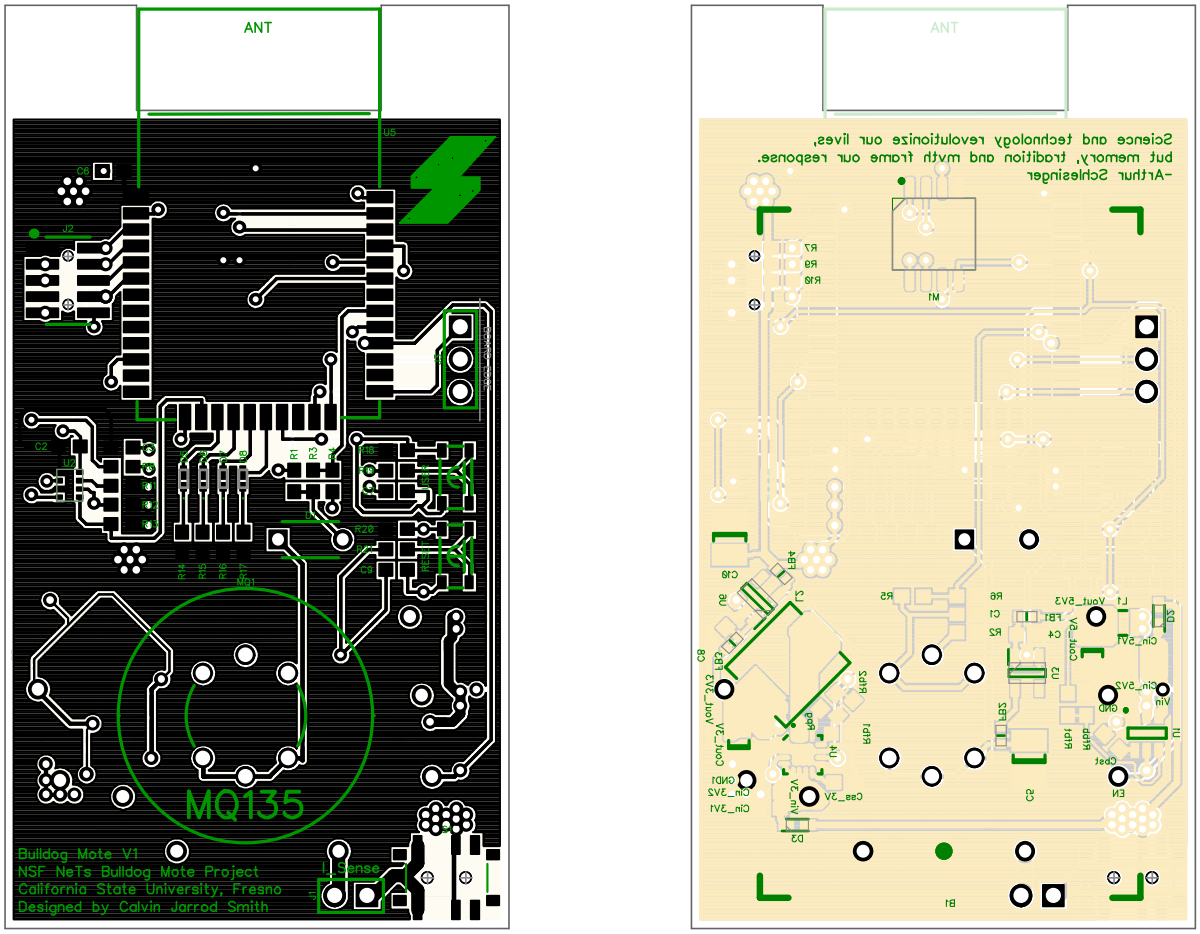


Figure 38: PCB Layout of Bulldog Mote.

The board sensors, JTAG, buttons and LEDs were all placed on the front side of the board, the power supplies and memory module were then placed on the back of the board, along with the battery pack. The large inductors of the power supplies needed to be kept as far away from the antenna as possible to reduce the possibility of EM interference. As well, the PCB board edge was modified so that there was no board directly underneath the PCB antenna of the CC2538 module, which also means that no ground or power plane of the PCB was underneath the PCB antenna either, which would adversely affect the antenna performance.

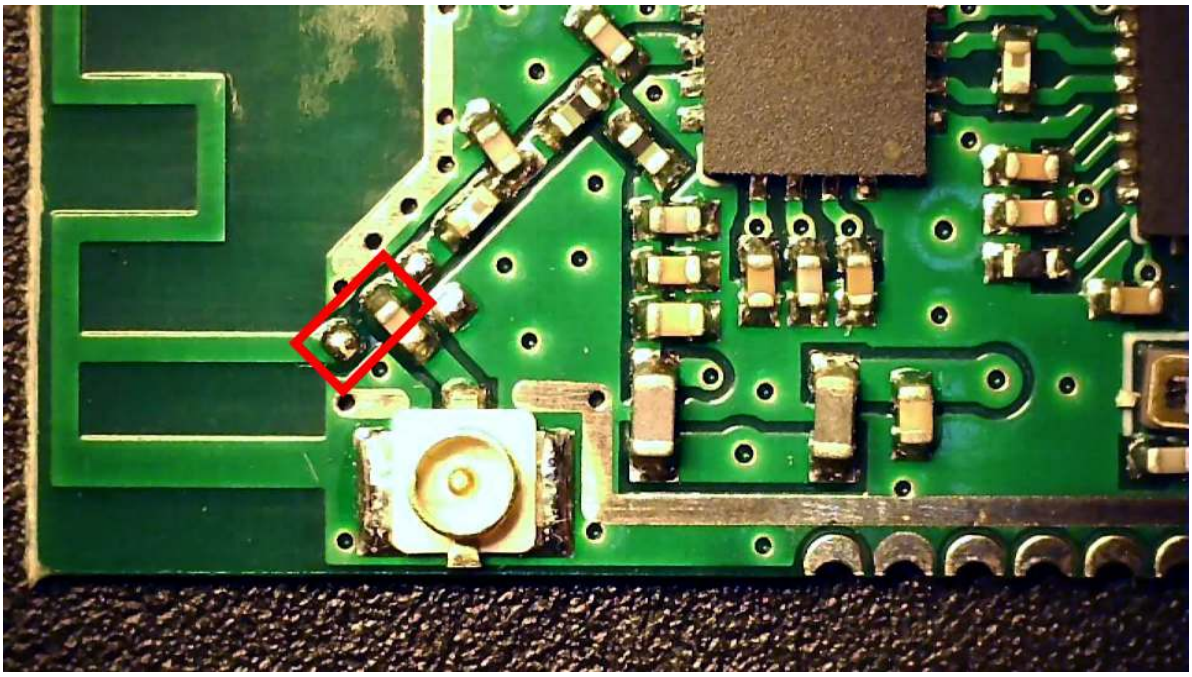


Figure 39: Inductor Connecting System to IPEX Connector instead of PCB antenna.

Upon receiving the CC2538 modules, it was observed that during assembly, the last inductor of the antenna balun was connected to the IPEX connector as opposed to the PCB antenna. On each module that was mounted to the Bulldog Mote system, this inductor had to be removed and resoldered to connect the PCB antenna instead.

5.5 Testing Procedure

Several tests were performed on the assembled Bulldog Motes. First was to test the DC voltage to ensure correct output voltage to both power systems. Then, the antenna was tested using the TI SmartRF Studio to ensure the replaced inductor did not adversely affect RF transmissions. Lastly, the Bulldog Motes were placed in a network and all sensor data was transmitted to a network sink to ensure all sensors were operating correctly. As well, the output of each power supply was analyzed through an oscilloscope before and after each LDO to ensure voltage ripple reduction.

The Bulldog Motes were connected to a DC power supply set to 9.0 V, just like the SM14 Shields, through the 9 V battery pack pins. The device was turned on using the power switch and the 3 V power supply output was measured using a multimeter.

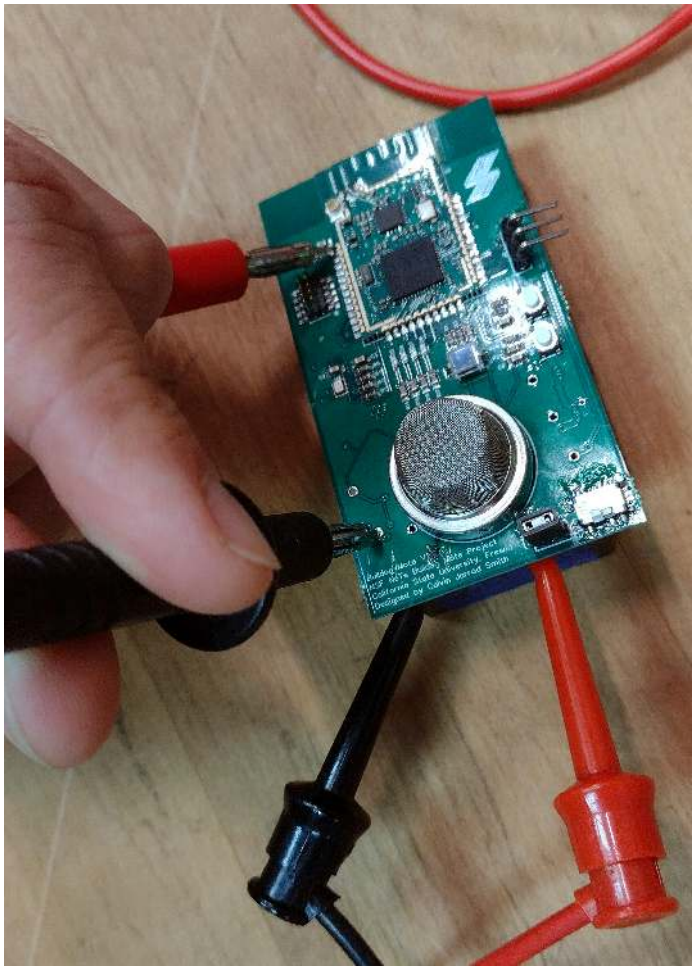


Figure 40: Testing the 3 V Power Supply of Bulldog Mote.

To test the 5 V power supply, the *enable* pin of the regulator had to have 3 V applied to it. Once 3 V was applied, the 5 V supply was measured. Then the enable pin was connected to 0 V and the supply was measured again to ensure the power supply correctly turned off.

The Bulldog Motes were left connected to the DC power supply, then the 3 V system output was measured using an oscilloscope. The oscilloscope was set to AC coupling with the 20 MHz bandwidth limit enabled to filter out unwanted environmental noise. As well, the ground lead of the oscilloscope probe was replaced with a ground spring to eliminate addition noise caused by the ground lead acting as an antenna.

Figure 41: Replacing the Ground Lead on Oscilloscope with Ground Spring.

The voltage ripple of the 3 V power supply was measured before and after the LDO on the board's test points. As well, the enable pin of the 5 V power supply was stimulated and the 5 V power supply was measured before and after its LDO on the board's test points.

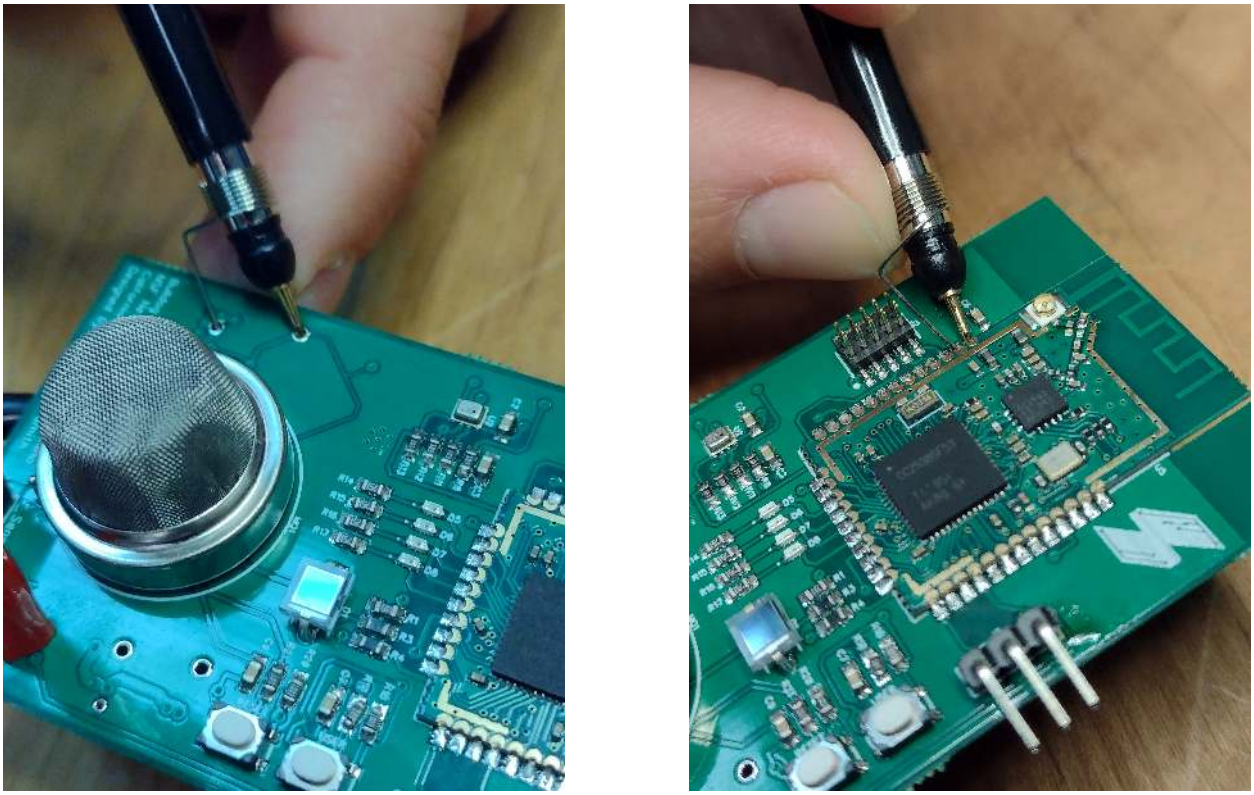


Figure 42: Test Points for the 3.3 V Switching Regulator (Left) and for the 3 V LDO (Right).

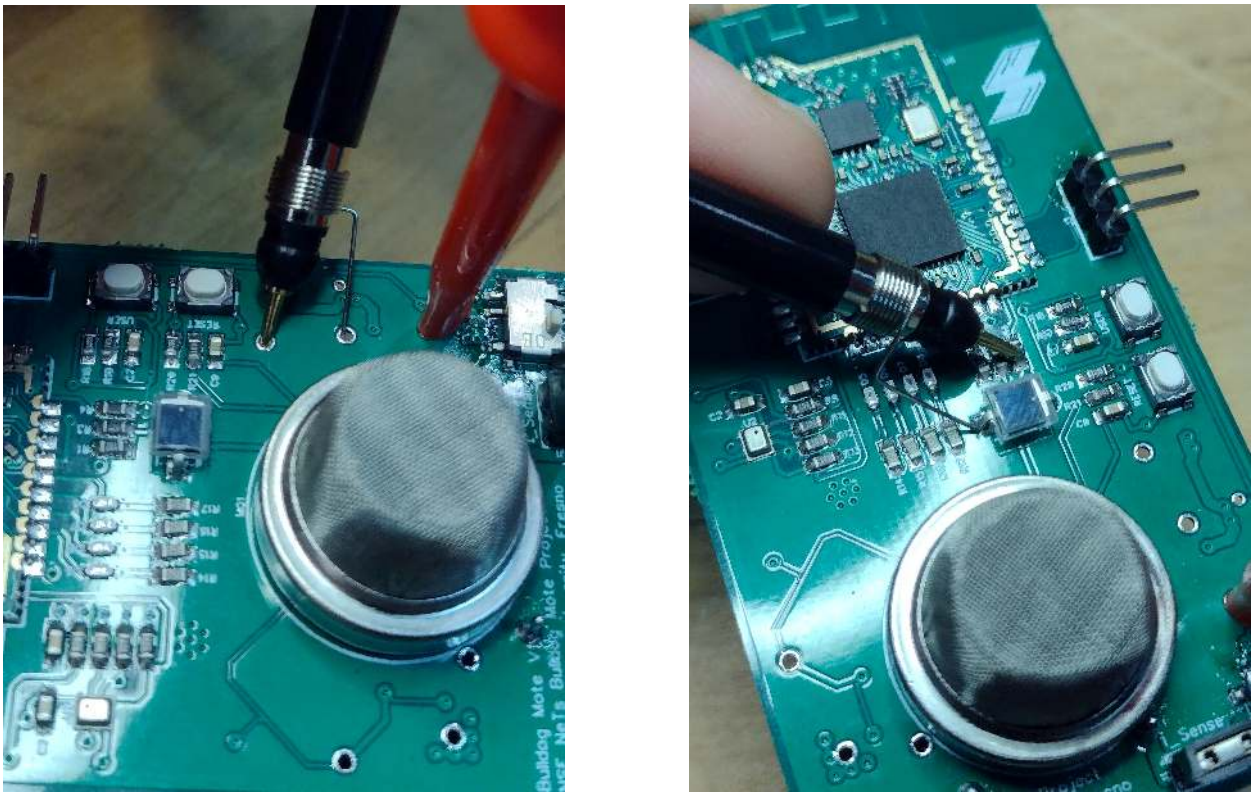


Figure 43: Test Points for the 5.3 V Switching Regulator (Left) and for the 5 V LDO (Right).

Next, the Olimex JTAG programmer was connected to the board through a 20-pin JTAG to 10-pin ARM JTAG adapter.

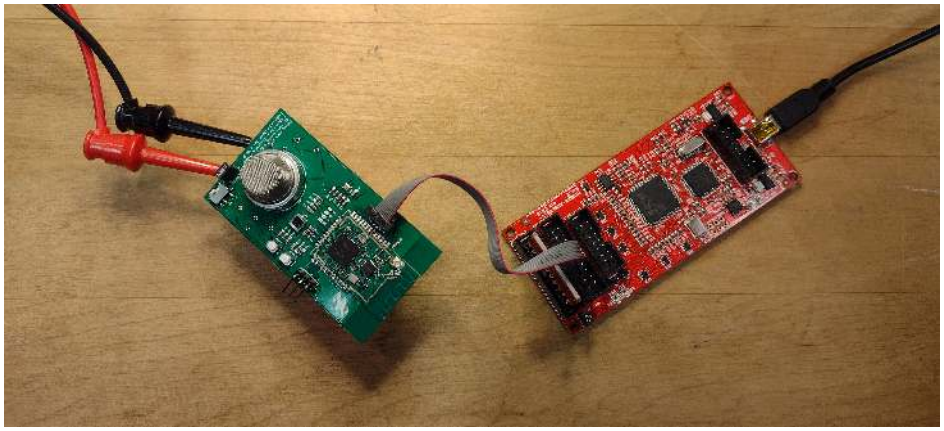


Figure 44: Bulldog Mote connected to JTAG Programmer.

The next test was for the antenna. The TI SmartRF Studio was started and used to test the board identically to the SM14 Shield system. The results of the test are shown in table 4.

For testing the sensors and networking ability, an application was written in Contiki NG.

The pinouts and devices on the Bulldog Mote were the same as the SM14 Shield, therefore the same testing program could be used for the Bulldog Mote. A TelosB Mote was used as a network sink attached the a host device to read-out the data being sent through the network.

5.6 Results

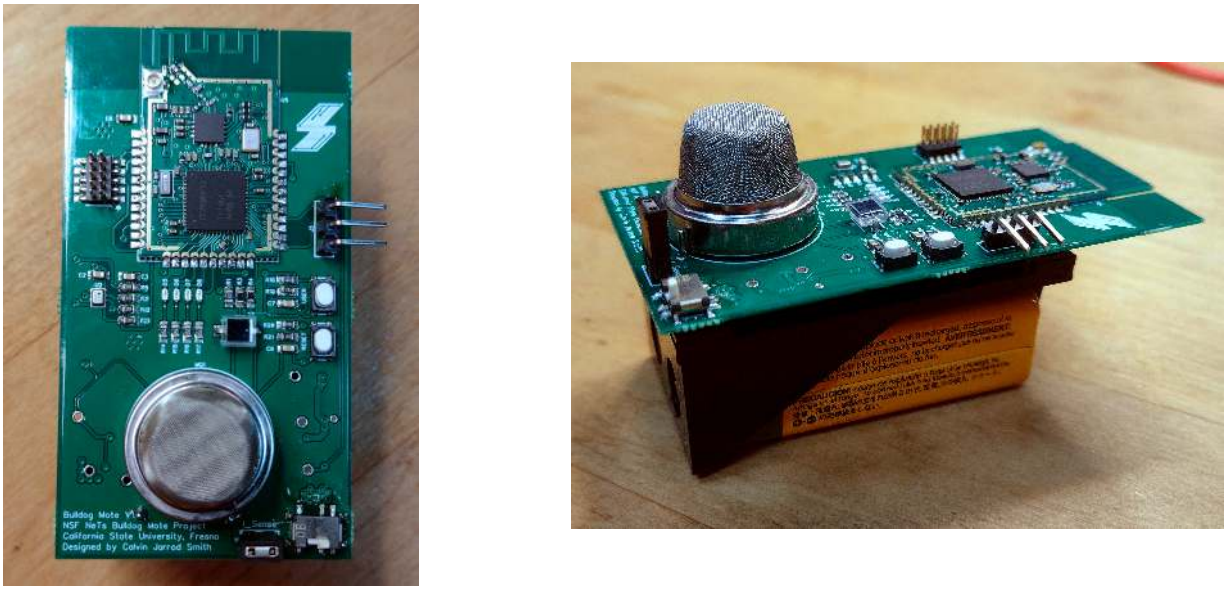


Figure 45: Completed Bulldog Mote.

The Bulldog Motes were tested to ensure all DC power supplies worked correctly. All 9 Bulldog Motes measured correct voltages at the output of each supply within 10% of desired voltage.

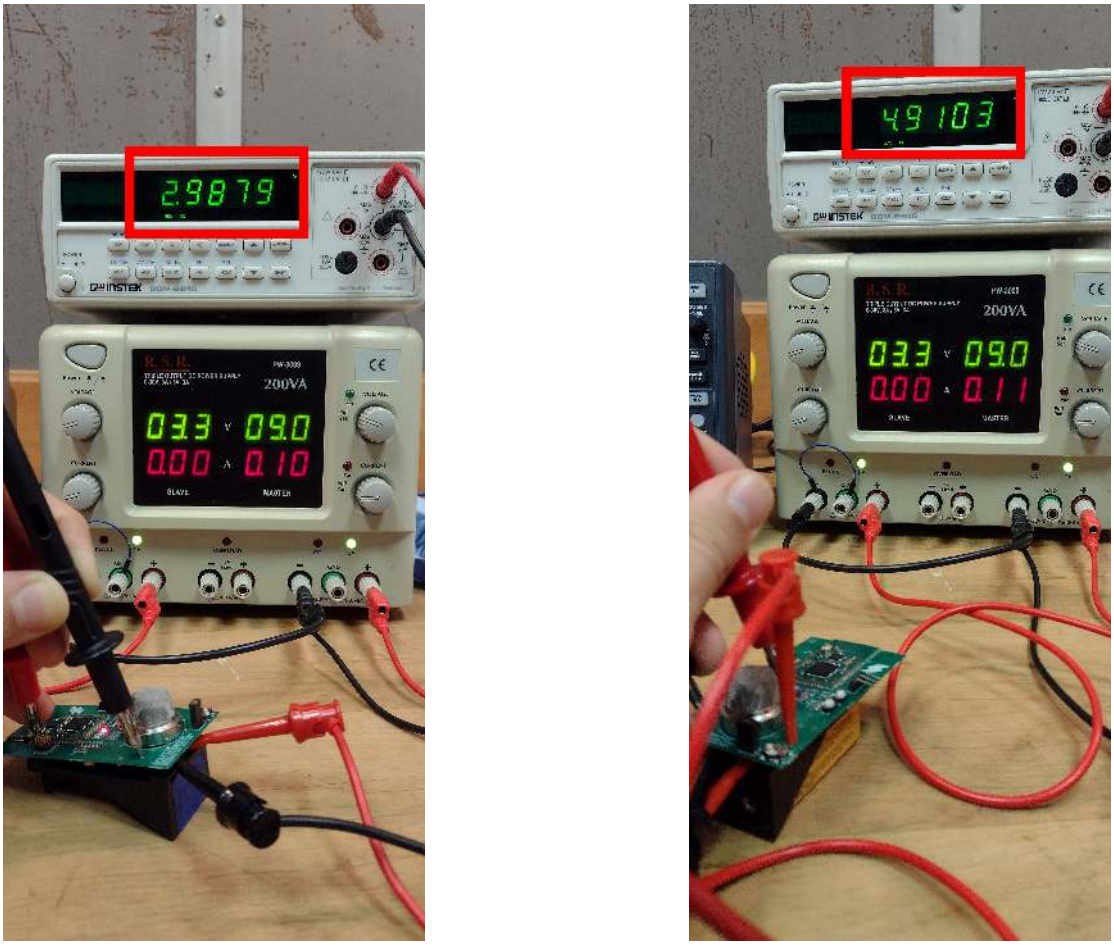


Figure 46: Output DC Voltage of 3 V System (Left) and 5 V System (Right)

In addition, an oscilloscope was used to measure the voltage ripple from each switching regulator output and compared to the voltage ripple after the LDO and ferrite bead filter.

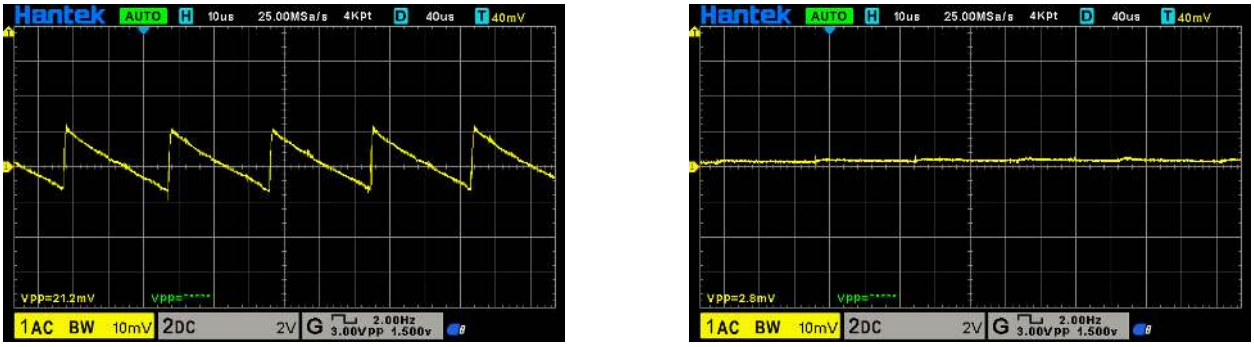
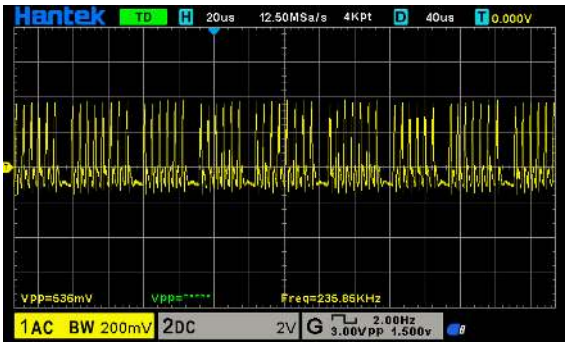


Figure 47: Voltage Ripple of 3.3 V Switching Regulator(Left) and 3 V LDO output (Right).

From figure 47, the attenuation of the ripple out of the switching regulator is -8.79 dB. This is an almost 10-fold reduction in ripple voltage, and a much more acceptable ripple of only 2.8 mV at the 3 V power system output. The same was done for the 5 V power system:



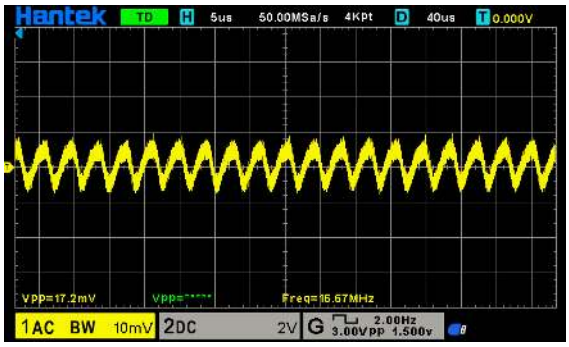


Figure 48: Voltage Ripple of 5.3 V Switching Regulator(Left) and 5 V LDO output (Right).

This system attenuated the voltage ripple by -14.94 dB, an even larger reduction than the 3 V system. The voltage ripple out of the 5 V system is very large, however since the reduction in the ripple through the LDO is significant and the 5 V supply is separate from the more sensitive 3 V system used to power the microcontroller, the output ripple is acceptable.

During the next test, the antenna circuits of each mote the following results were obtained while running the TI SmartRF Studio:

Mote RF Testing			
Mote Num.	**Total Packets**	**Packets Received OK**	**Packets Not Received OK**
1	100	99	1
2	100	72	1
3	100	97	1
4	?	?	?
5	100	100	0
6	100	99	1
7	100	98	0
8	100	99	1
9	100	90	2

Packets Dropped	**Packet Error Rate**	**Avg. RSSI (dBm)**	**Potential Issue**
0	0.01	-78.7	
27	0.28	-70.4	Ant. inductor
2	0.03	-90.2	
?	?	?	Faulty Module
0	0	-84.7	
0	0.01	-87	
2	0.02	-87.6	
0	0.01	-82.4	
8	0.1	-84.4	

Table 4: RF Testing Results of Bulldog Mote.

The criteria for passing a Bulldog Mote was to have 90% of transmitted packets received OK. Motes 2 and 4 did not meet this criteria. Mote 2 was fixed by resoldering the antenna inductor and reapplying the test, resulting in a 98 packets out of 100 being received OK, a 2% packet error ratio and an average RSSI of -83.8 dBm. However, several attempts were made to fix mote 4. On this mote, the inductor antenna was resoldered as well as the entire CC2538 module replaced. Neither of these attempts resulted in any readings from the SmartRF software and Mote 4 was thrown out. Out of the 9 motes, 8 passed this test after the antenna components were resoldered.

The sensor and network test were combined into one large test and the results were observed on the output of the serial readings of the TelosB mote that was setup as a UDP server. 4 out of 8 of the motes read a temperature of 0.0 degrees Celsius, which identified them as faulty. These 4 devices had the BMP280 temperature sensors replaced and had fully functioning temperature readings. The ALS and MQ135 sensors gave various, but consistent readings. Again these analog sensors require additional calibration to read an accurate lux or AQI measurement.

As a note, another benefit of using the external flash memory module can be to store the MQ135 and ALS sensor calibration data. The BMP280 sensor contains its own calibration data stored on the chip itself, that is needed to obtain correct readings of temperature and pressure. However, the BPW34 and MQ135 are analog sensors with no digital storage. So the flash memory could be used to store each mote's calibration data on an individual bases using a custom calibration application, then this data can be accessed when another application is programmed onto the board to used these sensors.

Out of the 9 Bulldog Motes assembled, 8 successfully passed. More motes can be produced in the future, however due to the current chip shortage in 2020 and 2021, the initial ICs for the power supplies were no longer in stock. Either the power supplies will need to be redesigned using different parts and retested, or the current design can be kept and more of the same parts ordered when supplies are replenished.

6 Lightning Strike AODV (LS-AODV) Algorithm

6.1 Overview

Ad-hoc on-demand distance vector (AODV) is a very well established routing algorithm in MANETs and WSNs. This algorithm is relatively simple, and helps deal with the issue of routing through a changing network and does not require constant routing maintenance [1]. Despite the prolific use of AODV, it lacks any awareness of energy consumption throughout the network. Lightning Strike AODV (LS-AODV) was proposed to help remedy this issue. The idea of LS-AODV was derived from the observation that routing *pathways* were being created as data traveled from source to destination in a mesh network. These pathways were then exploited to determine the path of highest energy reserve. Once found, the sending and receiving nodes communicate along this pathway, helping distribute the energy consumption of the network more equally, and therefore extending the network lifetime. A conference paper was written for this algorithm and accepted to IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference 2021.

6.2 Algorithm Development

6.2.1 Ad-Hoc On-Demand Distance Vector

In MANETs, constantly maintain routing information can drain a devices resources and battery life. In AODV, routing information is not kept fresh, in that maintenance message are not used to ensure a route still exists. This can be both good and bad. For energy constrained devices, this helps to save energy from constant radio transmission and allows longer sleeping periods. However, if the network requires real-time, or close to real-time responsiveness this lack of maintenance can cause delays that are unacceptable. Since WSNs and MANETs are typically more concerned with energy conservation this is a safe algorithm, despite the slower response [1].

When first beginning a transmission, a node must find a path to the network sink. Typically, this will cause a node to look through its routing table to find an entry for the destination node, and a resulting neighbor will be identified as the next hop to get there. However, if no entry exists the node will enter an *route discovery* phase, which begins the process of finding the destination node. This source node will begin by transmitting route request messages (RREQs) to all its neighbors. The node's neighbors in turn, check their routing tables for an entry to the requested destination. If a neighbor has an entry and it is considered *fresh* enough, the neighbor replies with a route reply (RREP) message back to the source node. This response message is then received by the source and it stores the new routing information in its routing table. Conversely, if none of the neighbors contain routing information to reach the requested destination node, they will also broadcast RREQs to their neighbors to try to obtain the routing information. This process continues until either the destination node is reached, or a node that has a fresh entry to the destination. In addition,

every node that receives a RREQ also stores a route to the source node in its routing table.

RREPs are handled using *unicast*, a single point-to-point transmission from one node to another. Once a route is determined by the destination node or one with a fresh route, a RREP is generated and transmitted to the neighbor through which the RREQ was received. In the case of many RREQs reaching a single node, the first RREQ is generally responded to with a RREP. Since a fresh route then exists after the first RREQ to the source, all others are dropped. The same is the case when RREQs are being rebroadcasted, and neighbors receive each other's RREQs from the same source. The RREQ sequence numbers and hop count are used to keep these packets from causing infinite rebroadcasts throughout the network.

Routing errors also handled dynamically, so a source node can delete its current routing information to a destination and begin the route discovery phase again. A route error (RERR) packet is generated once a node knows a path to a destination no longer exists. This packet is then sent back up the stream of nodes until it reaches the original source node [1].

6.2.2 Energy Aware-AODV (EA-AODV)

The AODV algorithm has not awareness of energy levels of nodes in a network, let alone make routing decisions based on it. In the paper titled "Energy Aware AODV (EA-AODV) Using Variable Range Transmission", Nayek et al. [2] introduced an extension on AODV that takes into account the transmission distance between nodes and adjusting transmission power as a way to save energy. The algorithm is broken down into several steps for transmission:

1. When a source node needs to send data and doesn't have routing information to destination, it broadcasts a RREQ with a transmission distance of 250 m (based on Friis free-space equation).

2. RREP messages contain two fields, $locX$ and $locY$, that contain the Cartesian coordinates of the sending node.

3. Upon reception of a RREP, a node must wait T_{wait} to receive all RREP that are destined for it.

4. Once T_{wait} has elapsed, the node calculates the distances to each neighbors it has received a RREP for a certain source and destination, it chooses the neighbor with the shortest distance to store as the next-hop node for the routing table entry used for the destination.

5. The routing table entry is also updated with the shortest-distance neighbor's X and Y position: n_{hopX} and n_{hopY}.

6. The transmission power of the node receiving the RREP is set based on the distance to closest neighbor using the Friis free-space loss equation. The threshold of received power is set constant based on wireless protocol.

7. The source to destination route is maintained for data transmission.

8. If a broken route occurs, repeat from step 1.

6.2.3 Need for Energy Balancing

The problem with EA-AODV is that it looks at energy savings from a per-node perspective. This doesn't allow network-wide routing to be changed based on energy consumption in certain areas. To remedy this problem, LS-AODV was devised. LS-AODV aims to distribute energy consumption more evenly throughout the network by identifying *energy pathways*. In AODV certain routing paths are determined by the distribution of RREQ messages. When a neighbor that has already received a RREQ with the same sequence number, source address and a greater or equal hop count, the packet is dropped. This creates distinct routing paths from source to destination. However, in AODV, the first RREP message to reach the originating node is the next-hop neighbor chosen for transmissions. With LS-AODV, the paths created by the RREQ, such as the ones shown in figure 49.

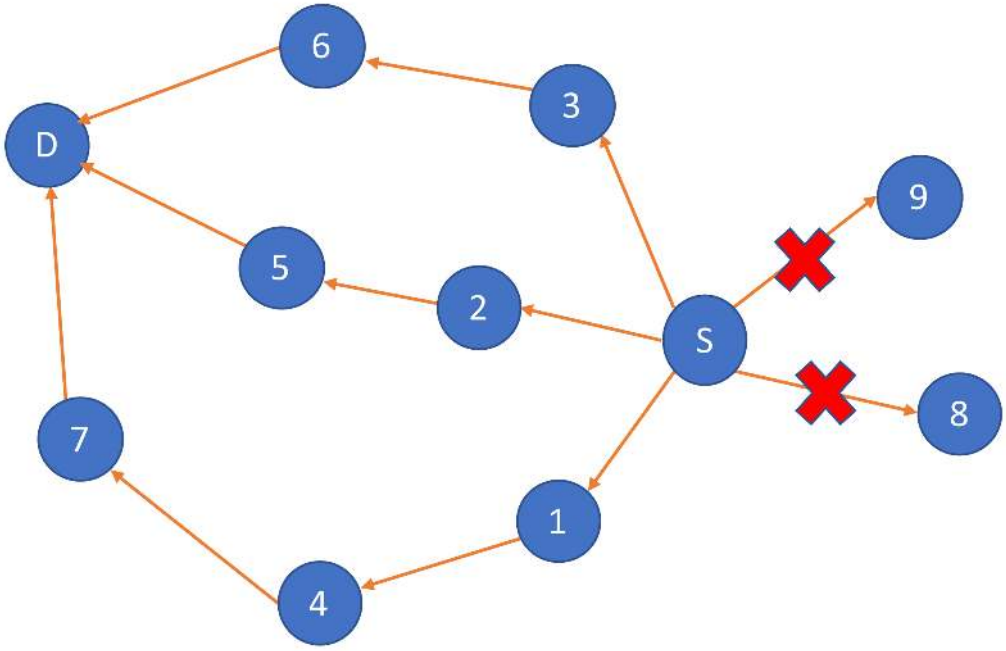


Figure 49: Potential Route Request Paths to Destination.

The algorithm is device as such:

1. RREQ messages contain a *energy accumulation* field that contains a 32-bit floating-point value, and a *current node energy level*.

2. Anytime a RREQ is encountered by a node that contains a routing table entry for the originator:

(a) with a sequence number greater than the one currently stored: the current node's energy level is added to the RREQ's *energy accumulation* field and placed in the current node energy level field before processing and forwarding.

(b) with an equal or lesser sequence number: the RREQ is dropped.

(c) updates its neighbor's energy level stored in its routing table in the entry for that neighbor.

3. Nodes originating a RREQ insert only is own energy level in the the energy accumulation field and the current node energy level field.

4. When a node receives a RREQ that is destined for it, the node saves the RREQ and waits T_{wait} for additional RREQs to reach it.

5. After T_{wait} has elapsed the node calculates the *average energy per node* by dividing the energy accumulation field of each RREQ by the RREQ hop count plus 1. The RREQ that results in the highest average energy per node is the one chosen for the RREP message response.

6. Similarly to EA-AODV, RREPs contain a node's X and Y coordinates. In addition, RREQs also contain a transmitting node's X and Y coordinates. These coordinates are stored in the routing table data for source and destination nodes.

7. All transmitted packets are sent by adjusting antenna transmission power to the distance based on the X and Y coordinates stored in the routing table using the Friis Free-Space equation and a minimum receive power of

As an example, the same network topology was used from figure 49. Each node is given the initial energy of the values from table 5

Node Number	Initial Energy at time RREQ is received (J)
1	87
2	59
3	27
4	80
5	54
6	34
7	77
8	23
9	64

Table 5: Initial Energy of Nodes in Joules.

Figure 50 shows node S broadcasting a RREQ to reach node D. Similar to AODV, nodes that do not have a routing table entry for device D will also broadcast RREQ messages to their neighbors. These nodes will also store an entry for S along with its coordinates and

current energy level. Nodes 1, 2, 3, 8 and 9 that receive the first RREQ will then broadcast another RREQ with the same originator sequence number and address, but now with their own energy value added to the RREQ accumulated energy field.

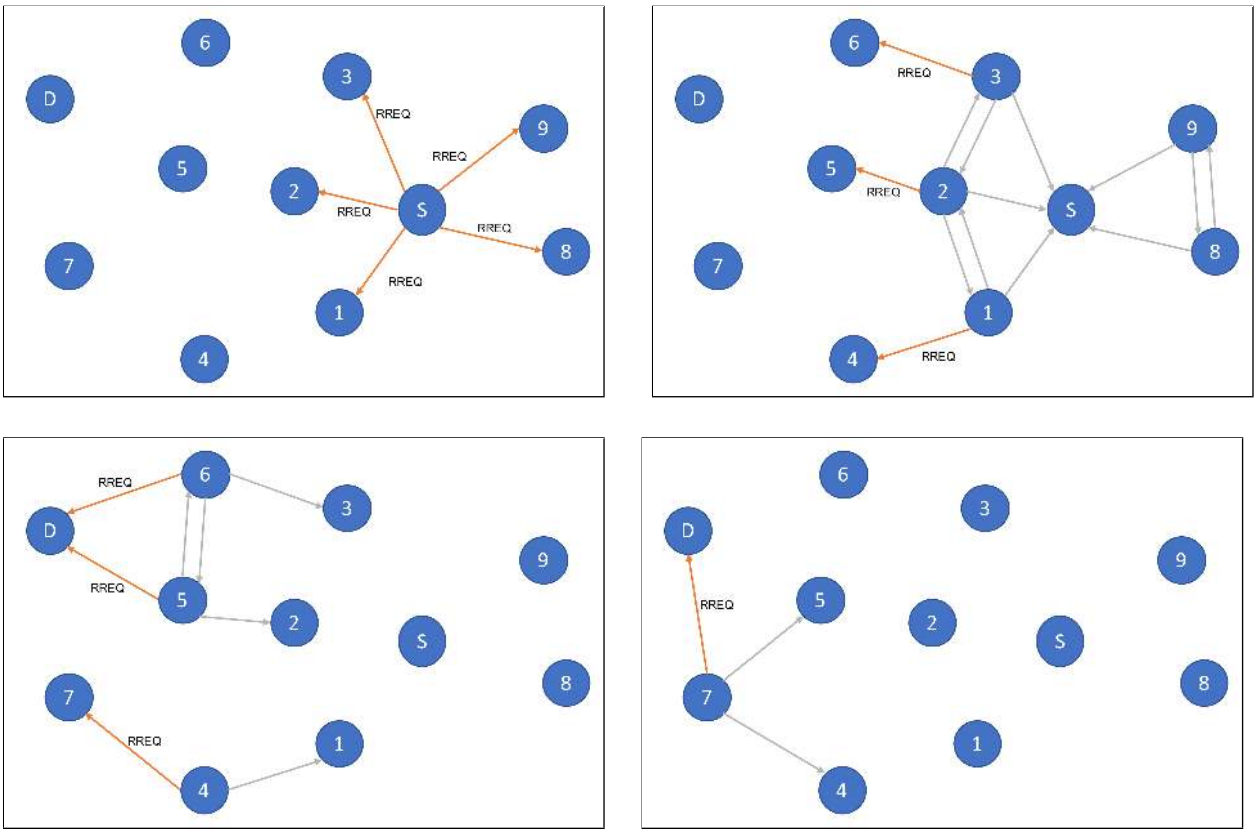


Figure 50: RREQs broadcasted from Node S toward Node D.

RREQ propagation occurs identically to AODV, with the addition of each node adding its own energy level to the two fields of the RREQ packet. Once node D receives the first RREQ from S, it will start a timer for the specific sequence number and source address combination. Assuming that node 6 obtains the RREQ from 3 first, 5 from 2 first, and 4 from 1 first, then 7 from 4 very shortly afterward, three distinct routing paths shown in figure 49 will exist. The RREQ received at D from 6 will contain an accumulated energy value of $27 + 34 = 61$ J while hopping from $S \rightarrow 3 \rightarrow 6 \rightarrow D$. The accumulated energy from 5's RREQ propagating through $S \rightarrow 2 \rightarrow 5 \rightarrow D$ would be $59 + 54 = 113$ J and from 7 through $S \rightarrow 1 \rightarrow 4 \rightarrow 7 \rightarrow D$ would be $87 + 80 + 77 = 244$ J.

Once D receives all of the RREQs, it will take these values and divide by the RREQ hop count to determine the average energy level per node encountered up until that point. For RREQs received through 6, 5 and 7 the average node energy would be 30.5, 56.5 and 112, respectively. Node D will chose the path back to S through node 7 as the average energy per node is higher, despite the fact that the hop count is greater and most likely took longer to be received than the other RREQs (Figure 51). This return path for RREPs determines

the route in which S will transmit to D for future transmissions.

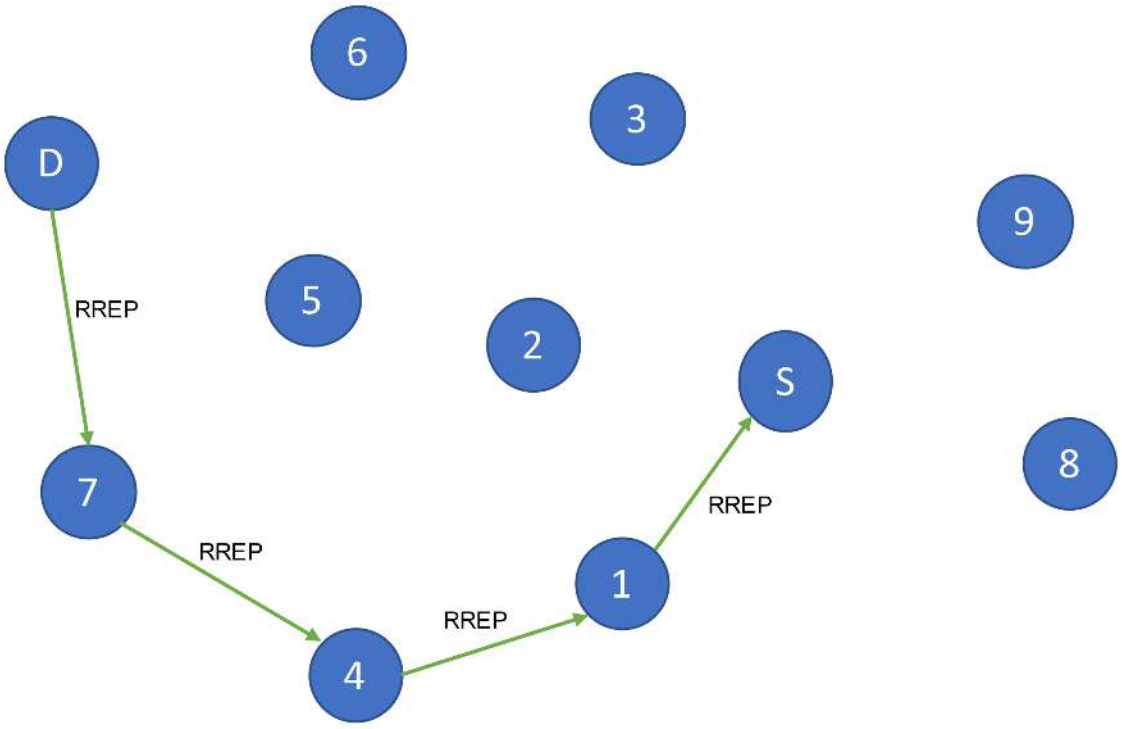


Figure 51: RREP Return Path from D to S.

Lastly, both the RREQ packets and the RREP packets contain the X and Y coordinates of the sending node. These coordinates are stored as part of a routing table entry for each node's neighbor. Energy savings occur for all unicasted messages (RREP, RERR and data transmission) in LS-AODV by reducing transmission power to just reach the node which the packet is destined for. Broadcasted messages (RREQs) are set to a default transmission range since RREQs are potentially used for the discovery of new neighbors in MANETs.

6.3 Simulation

6.3.1 NS-3

Simulations for LS-AODV, EA-AODV and AODV were performed in *ns-3*, a discrete-event driven network simulator [16]. *ns-3* has been used for research and academic purposes and is capable of handling wireless and traditional networking interfaces.

6.3.2 Energy Model

A simple energy model was created based in each routing algorithm to aid in monitoring energy consumption. *ns-3* already contains a more complex energy model with energy

harvesting and specific battery types, however, they require device current to be known. However a simpler energy model that was integrated into the routing protocol was used.

Each routing protocol was modified with an variable for energy level in Joules and made accessible using the *ns-3* Attribute System. This allows modules in different levels of the compilation hierarchy to be accessible to each other.

Wireless sensor devices have several sources that can deplete battery life [17]:

1. Wireless transceiver receive (RX) energy
2. Wireless transceiver transmission (TX) energy
3. CPU Active energy

Each one of these components will need to be considered for a proper working energy model.

Since all wireless microcontrollers (MCUs) have different power ratings and current consumptions based on the type of MCU, the wireless protocol and the various sleeping modes, a single MCU should be chosen based on some previously known factors. In *ns-3*, the algorithm that the energy model was installed onto was the Ad-Hoc On-Demand Distance Vector (AODV) algorithm. This routing model uses WiFi protocol for wireless transmission. Knowing this, the MCU that was chosen was the ESP32-WROOM32 device for its low-power WiFi transmission capabilities (Figure 52). This MCU was also very prolific in terms of WSN connectivity with WiFi protocol.

Figure 52: ESP32-WROOM-32 Embedded Wireless MCU.

For this device, the CPU and transceiver power characteristics were obtained from the ESP32 datasheet on Espressif's website [18]. In Table 15 of the datasheet, the TX and RX current for the ESP32 device for the IEEE 802.11b protocol was 240 mA and 97.5 mA respectively, including CPU contributions. In Table 6, the CPU current consumption when in *Modem-Sleep* mode (without RF) is on average 22.5 mA for the single-core at 80 MHz. For this project, only the active mode was considered for the sake of simplicity. It was mentioned in the note below the table that the ESP32 switches between *Active* mode (with RF from Table 15) and *Modem-Sleep* mode. Since we will be assuming the transceiver will be constantly on for receiving incoming data, the contribution of just the wireless hardware

had to be found. This was done by taking the *Modem-Sleep* current usage out of the total current draw when the receiver is on [18]:

$$I_{RX} = I_{total} - I_{Modem-Sleep}$$

So the RX current for this protocol was found to be on average 75 mA. This along with the active, modem-sleep mode was used to determine the current expenditure while receiving. Wireless transmissions could have been handled in two different ways: use the datasheet values for transmission with a set propagation distance, or use a variable propagation distance model using standards for IEEE 802.11b. This project followed the latter option using the energy model from [2] and the Friis propagation loss equation.

Friis Propagation Loss Equation

For some applications, adjusting transmission range can help reduce energy consumption when sending data. To facilitate this feature, the Friis transmission equation can be used, assuming free-space propagation, to determine power usage based on transmission distance:

$$P_r = \frac{P_t G_r G_t \lambda^2}{(4\pi d)^2} \tag{11}$$

Where P_r and P_t are respective power of the RX and TX, G_r and G_t are the respective gains of the RX and TX antenna, λ is the wavelength in meters, and d is the distance between the sender and receiver in meters [19].

In equation 11, the respective antenna gains can be assumed to be unity for simple, low-power MCUs and the equation can be rearranged to find the TX power:

$$P_t = \frac{P_r (4\pi d)^2}{\lambda^2} \tag{12}$$

The IEEE 802.11b MAC protocol standard sets a ceiling to the minimum RX sensitivity level of -76 dBm, however chip manufacturers can specify less than this level as long as the frame error ratio (FER) does not fall below 0.08 for 1024 bytes [20]. The ESP32 device has its own sensitivity level for receiving data of -98 dBm for the 802.11b standard with 1 Mbps [18]. This value was used as the RX power, P_r, to maximize energy savings by reducing needed output power to a minimum. In addition, the wavelength for WiFi protocol at 2.4 GHz can be found using:

$$\lambda = \frac{c}{f} = \frac{2.998 \times 10^8 \ m/s}{2.4 \times 10^9 \ Hz} = \boxed{0.125 \ m}$$

Resulting in a power equation of:

$$P_t = \frac{10^{\frac{-98-30}{10}} \cdot 16\pi^2}{0.125^2} \cdot d^2 = 1.6018 \times 10^{-9} \cdot d^2 \tag{13}$$

However, this equation was in units of power, not energy. So to deplete a battery source, the power equation had to be translated to energy. To do so, the bit transfer rate of 1 Mbps was used to help determine the time it takes to transmit data of a certain length:

$$E_t = \frac{\#\ bits}{1 \times 10^6\ bps} \cdot P_t = \boxed{1.6018 \times 10^{-15} \cdot (\#\ bits) \cdot d^2} \tag{14}$$

Equation 14 shows the energy usage for a single transmission of a certain number of bits at a variable distance. So within the *ns-3* code, the value $1.6018 \times 10^{-15} \cdot (\#bits)$ was saved as a constant within the routing protocol code, as the packet size in bits was known ahead of time.

For depletion of active CPU and RX energy, the current consumption was used as well as the ESP32 input voltage for V_{DD}, which was 3.3 V, to find power:

$$P = IV$$

This power value can then be used to find energy consumption of a battery in a similar way as the TX energy usage:

$$E = Pt \tag{15}$$

Where t is time in seconds. The current values for RX (75 mA) and active CPU (22.5 mA) were included as constant values in the routing protocol in *ns-3*. As well, a periodic battery decay function was implemented and executed every 2 ms to decrement the battery energy. These calculations were used while programming each routing protocol, as shown in the next section.

6.4 Implementation

To incorporate a working energy model in an existing routing algorithm in *ns-3*, the routing protocol needs to be modified. The AODV routing protocol is located in the {`NS3_-HOME`}`/src- /aodv/model`. Within this directory both the `aodv-routing-protocol.cc` and `aodv-routi- ng-protocol.h` were modified.

Routing Protocol Header

In the top of the routing protocol header, two constants were defined above the class definition:

```
#define        PACKET_SIZE        64
#define        ESP32_VOLTAGE    3.3f
```

These were declared above the class to allow global access to the values. Next, within the `RoutingProtocol` class, the following static variables were included:

```
static const float power_receive;
static const float power_active;
static const float m_aliveThreshold;
static const double TxConst;
```

These variables were the ones discussed previously, along with an *alive threshold* which indicates the minimum energy level a node can have while still being considered "alive". All of these variables were declared as *static constants* because they were the same for all nodes and only one instance of these variables for all `RoutingProtocol` objects was more efficient for the simulation workload.

Declared in the *public* access of the class were included:

```
double m_energyLevel;
double transDistance;
Ipv4Address ipv4Addr;
```

These were used for the device energy level and the transmission distance set by the simulator, plus the node's IPv4 address. A decay function needed to be called periodically to simulate active CPU current and RX transceiver current load on the battery. An *ns-3 Timer* object, along with a timer callback function were declared in the *private* access to facilitate this:

```
Timer m_powerDecayTimer;
void PowerDecayTimerExpire();
```

Routing Protocol Source Code

Next within the `aodv-routing-protocol.cc` source code file, in the top of the `aodv` namespace, the energy model constants were initialized using:

```
const float RoutingProtocol::power_receive = 0.075 * ESP32_VOLTAGE;
const float RoutingProtocol::power_active = 0.0225 * ESP32_VOLTAGE;
const float RoutingProtocol::m_aliveThreshold = 0.001;
const double RoutingProtocol::TxConst=(double)(PACKET_SIZE*8)/1e6 * 1.601768e-9;
```

These values were determined in the *Theoretical Background* using the ESP32 datasheet and verified to be part of the IEEE 802.11b standard [18, 20]. The `power_receive` and `power_active` were set to values determined above multiplied by the ESP32 system voltage. The `m_aliveThreshold` value was set to 0.001 as a low, non-zero value that can mark the expiration of a node's battery. As well, the `TxConst` was initialized as the value shown in Equation 14. `PACKET_SIZE` was multiplied by 8 as it was in terms of bytes, not bits.

Next, in the `RoutingProtocol` constructor, default values for the energy level and transmission distance were placed in the initialization list:

```
RoutingProtocol::RoutingProtocol ()
  : ...
    m_energyLevel(1000.0),
    transDistance(250.0),
    ...
{...};
```

This sets the default energy level to 1000 J and default transmission distance to 250 m (which is the default transmission distance of a node with the Friis Propagation Loss Model installed in *ns-3*). These values are not modified in the constructor argument list because *ns-3* has an alternative means to modify object variables across models without objects needing to be aware of specific object types nor need them to be in scope. This is accomplished using the *ns-3 Attribute System*, which exposes *attributes* connected directly to object fields. These object variables can be changed by using simulator commands that modify the variables at the simulator level. To enable this feature, the variables plus accessor types need to be included in the `GetTypeId` function. This function sets the namespace, group name necessary for accessing the attributes, and the attributes themselves. At the end of the list of `AddAttribute` dot operations, another two were added for the energy level and transmission distance:

```
TypeId
RoutingProtocol::GetTypeId (void)
{
static TypeId tid = TypeId ("ns3::ebaodv::RoutingProtocol")
    .SetParent<Ipv4RoutingProtocol> ()
    .SetGroupName ("ebaodv")
    .AddConstructor<RoutingProtocol> ()
    .AddAttribute ("HelloInterval", "HELLO messages emission interval.",
    ...
    .AddAttribute ("MaxPower",
        "Maximum battery power of a node",
        DoubleValue(1000),
        MakeDoubleAccessor (&RoutingProtocol::m_energyLevel),
        MakeDoubleChecker<double> ())
    .AddAttribute ("TransDistance",
        "Transmission Distance for AODV algroithm",
        DoubleValue(250),
        MakeDoubleAccessor (&RoutingProtocol::transDistance),
        MakeDoubleChecker<double> ())
    ;
    ...
}
```

The last two `AddAttribute` dot operations made the `m_energyLevel` and `transDistance` variables accessible through the `MaxPower` and `TransDistance` attribute names. The first argument is a string with the attribute name, the next argument is a description of the attribute. *ns-3* also uses its own casting for primitives, so the default values for each attribute is passed through a `DoubleValue`. Then the configuration path to each variable was given in `MakeDoubleAccessor` for the simulator to access. Then the last argument was

a checker object for the `double` value: `MakeDoubleChecker<double>`. This addition made both variables visible to the main simulation function for customization.

The next modification is in the `DoDispose` function, which functions as a destructor for the `RoutingProtocol` object. *ns-3* has its own `Object` class from which all objects are inherited. This base `Object` maintains its own garbage-collection at the end of a simulation, so if things need to be done at the end of a simulation when an `Object` is terminated, those things need to be placed in the `DoDispose` class method. In this class, a simple output statement is used to indicate how much energy the node has left. So at the top of the function this was placed:

```
if (m_energyLevel >= m_aliveThreshold) {
    std::cout<<"Node "<<ipv4Addr<<" has "<<m_energyLevel<<" remaining energy."<<std::end
}
```

where `ipv4Addr` is the node's IPv4 address. This statement only prints when the node is still alive, hence the condition: `m_energyLevel >= m_aliveThreshold`. Later in the code a statement was placed for indicating when a node's battery expires.

The next function modified was the `RoutingProtocol::Start()`. This function is called when a `RoutingProtocol` instance is created and installed on a node. Rate limit timers were set in this function to limit the number of route requests (RREQs) and route replies (RREPs) a node sends out. This helps to limit congestion caused by route discovery in a network. At the end of the function these lines were inserted:

```
m_powerDecayTimer.SetFunction (&RoutingProtocol::PowerDecayTimerExpire, this);
m_powerDecayTimer.Schedule(MilliSeconds(2));
ipv4Addr = m_ipv4->GetAddress (1, 0).GetLocal ();
std::cout<<"Node created "<<ipv4Addr<<" with initial power "<<m_energyLevel<<std::endl;
```

This code segment uses the `m_powerDecayTimer` timer object and creates a periodic callback to the `PowerDecayTimerExpire` function to decay the node's energy level every 2 ms. As well, `ipv4Addr` was initialized with the node's IPv4 address using the routing protocol's `IPv4` protocol object. Then an output statement was used to output the node address and starting energy level.

To keep the node disabled when the energy level is equal to or below the alive threshold, conditional blocks were placed at the beginning of several functions:

```
if (m_energyLevel <= m_aliveThreshold) return;
```

This statement was placed in the top of the `RouteInput()`, `RouteOutput()` and `RecvAodv()` functions. This was because both `RouteInput()` and `RouteOutput()` are entry points to the routing protocol from the application layer. Also, the `RecvAodv()` is called whenever a route discovery is initiated by the current node *or* a another node. So placing a

conditional check at the top of this function keeps the current node from participating in a route discovery when the node's battery is dead.

The `PowerDecayTimerExpire()` function was included just under the `RerrRateLimitTimerExpire()` function. This was the function that decays the batter every 2 ms based on the active CPU power and RX power:

```
void RoutingProtocol::PowerDecayTimerExpire() {
    m_energyLevel -= (power_active+power_receive)*0.002;
    if (m_energyLevel < m_aliveThreshold) {
        std::cout<<"Node "<<ipv4Addr<<"  died at time "\
            <<Simulator::Now().GetMilliSeconds()<<"ms."<<std::endl;
        m_powerDecayTimer.Cancel();
    } else {
        m_powerDecayTimer.Schedule(MilliSeconds(2));
    }
}
```

The first instruction decrements the energy level based on the active power and RX power based on Equation 15 using 2 ms as the time duration. The conditional block checks if the node had just died in the current decay function call. If so, an output statement is used to tell the user that this node's battery has expired. As well, the timer itself is canceled since it is no longer needed. Otherwise if the node still has remaining energy, the timer is reset to 2 ms using `m_powerDecayTimer.Schedule(MilliSeconds(2))`. This takes care of the periodic energy decay of the battery.

The last part of the energy model is to emulate TX power decay. This is not done as simply as the RX and active CPU power decay as there are many places in the routing protocol and application layer that a transmission can occur. For the case of the routing protocol there are generally two places to decay TX power: in the `SendTo()` function, and every instance of `socket->SendTo()`. Within the `RoutingProtocol::SendTo()` a decrement statement was added:

```
void
RoutingProtocol::SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address destination
{
    // Energy Model
    m_energyLevel -= TxConst * powf(transDistance,2);
    socket->SendTo (packet, 0, InetSocketAddress (destination, EB_AODV_PORT));
}
```

The `m_energyLevel` operation was the addition to this function, where the energy level was reduced by the amount determined from Equation 14, since `TxConst` already takes into account the number of bits. The transmission distance was squared using the `powf` C++ standard floating point exponent function, so `<cmath>` was also added to the source code libraries.

Next, every instance of `socket->SendTo()` in the `aodv-routing-protocol.cc` file had to have the same line added before all of them as well:

```
m_energyLevel -= TxConst * powf(transDistance,2);
```

This was done using the `Vim` text editor's search function.

The difference between these two types of sending functions is the fact that some functions needed an indirect call to the `RoutingProtocol::SendTo()` function using a `Simulator::Schedule()`, which schedules the send function to run in the future. This can allow a small delay before executing the transmission by adding some jitter in transmission, like in real systems where queuing delays, operating system event management, or other phenomenon might delay the transmission temporarily. However, calling `socket->SendTo()` invokes the socket to send the data immediately with no delay.

These changes to the routing protocol included all battery decay elements: RX power, TX power and CPU active power and output statements to show when these operations occurred.

This new energy model was used to perform energy simulation of the LS-AODV, EA-AODV and AODV algorithms for comparison of energy savings, and network lifetime maximization. All source code created for this algorithm is listed in Appendix B.

6.5 Simulation and Results

Three network sizes were used during testing: 50, 75, and 100 nodes. For each of these three sizes, each node was randomly assigned using a normal distribution with a mean of 125 and standard deviation of 50. Each topology enclosed in a 250m x 250m square, where (125,125) was its center. Figure 53 shows the 100 node topology.

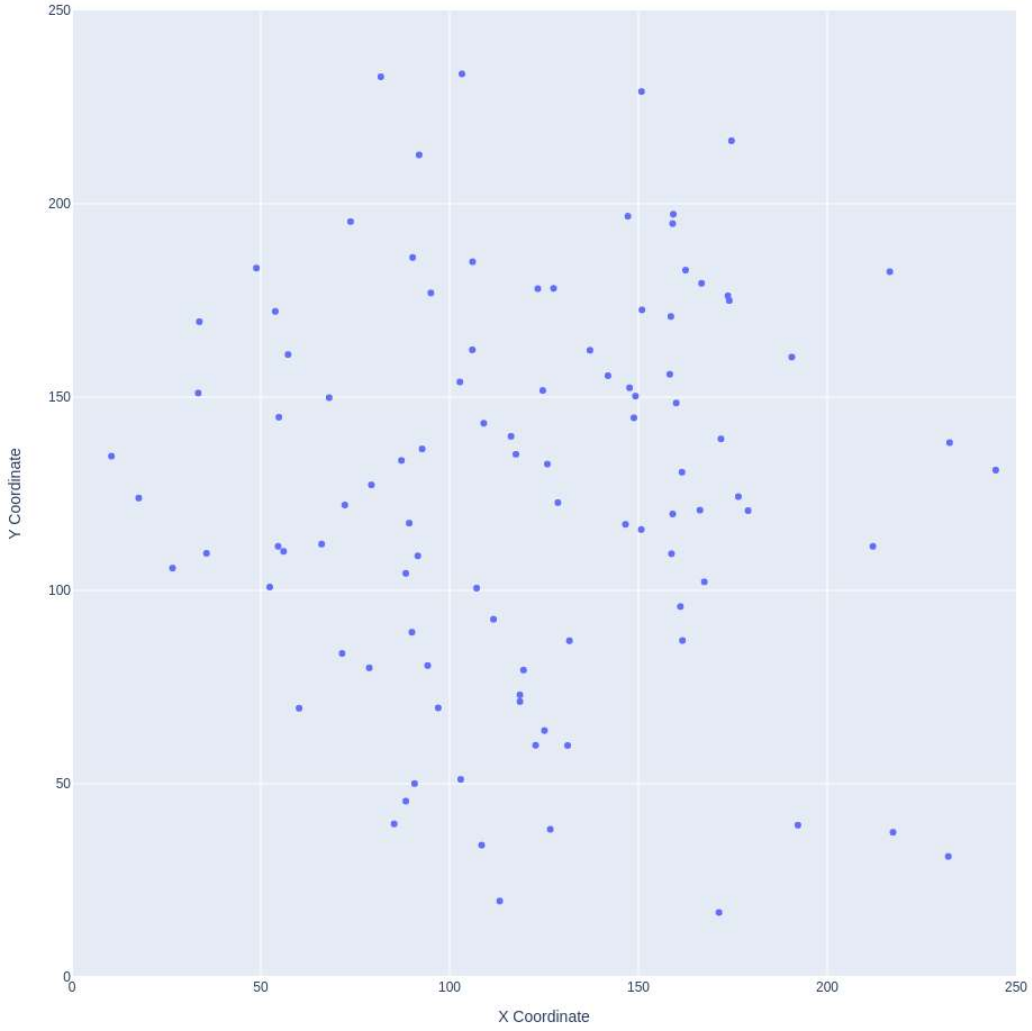


Figure 53: 100 node topology

Packet size was assigned as 4096 bytes and injection rates were adjusted for each simulation to 1, 2, 5, and 10 packets per second. Destinations of each packet were randomly selected. Each RREQ, RREP and RERR were 64 bytes. The 12 combinations of simulations with each combination of network size and injection rate were tested on LS-AODV and EA-AODV for comparison.

The energy model above using the ESP32 device as a reference was used to measure energy on each node. To start, each node bagin with 100 joules of energy, and each simulation ran for 10 simulated seconds. Nodes subtracted from their energy level based on transmission power, receiving power, and idle waiting. When a node reaches zero power, it was considered *dead* and no longer transmits nor participates in other node's route discovery. If a node cannot reach any other nodes due to all of its neighbors having died, it is considered *unreachable*.

Table 6 shows the remaining amount of energy in each simulation for both EA-AODV

and LS-AODV.

# Nodes and Injection Rate	Energy Remaining(J) EA-AODV	Energy Remaining(J) LS-AODV	% More Energy Remaining in LS-AODV
50, 1pps	3287.97	3190.36	-2.97
50, 2pps	2401.02	2940.24	22.46
50, 5pps	1613.94	17.43.07	8.00
50, 10pps	882.22	1179.92	33.75
75, 1pps	4245.48	4603.86	8.44
75, 2pps	3142.97	3441.26	9.50
75, 5pps	1557.52	2026.99	30.14
75, 10pps	1098.51	1377.25	25.37
100, 1pps	1785.94	2396.61	34.19
100, 2pps	4253.51	4443.51	4.47
100, 5pps	2361.38	3081.83	30.51
100, 10pps	2103.09	2523.06	19.97

Table 6: Energy Remaining in Networks.

Eleven of the twelve configurations show LS-AODV finishing with a higher amount of energy remaining in the network when compared to EA-AODV. Only the 50 node, 1 packet per second simulation shows more energy remaining in the EA-AODV network, with LS-AODV with 3% less energy. For the remaining twelve, most LS-AODV networks have a double digit percentage points improvement, with a best performance in the 100 node, 1 packet per second network improving by 34.19%. Figure 54 shows the results of the 100 node 1 pps simulation for EA-AODV, and figure 55 shows the results for 100 node 1 pps in LS-AODV.

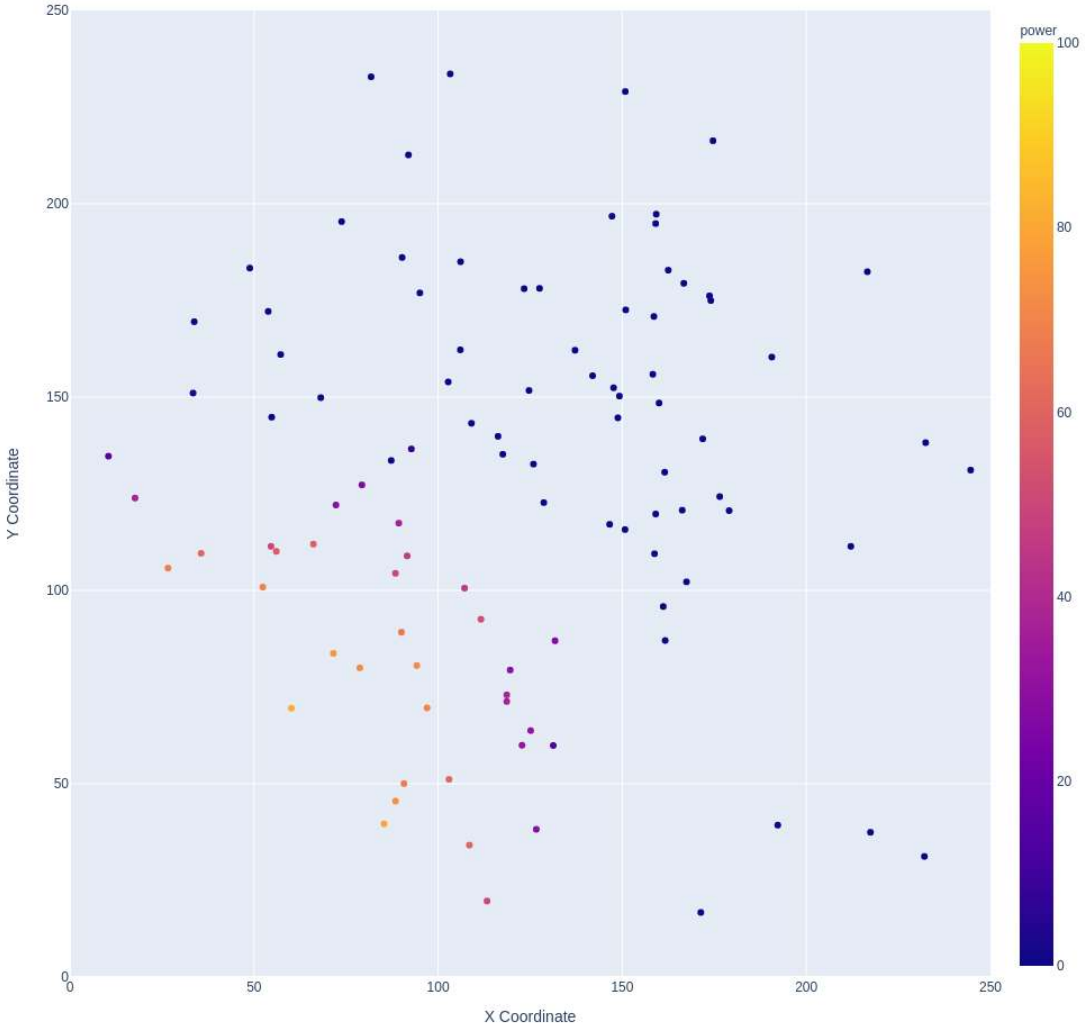


Figure 54: 100 Nodes 1 PPS EA-AODV Simulation

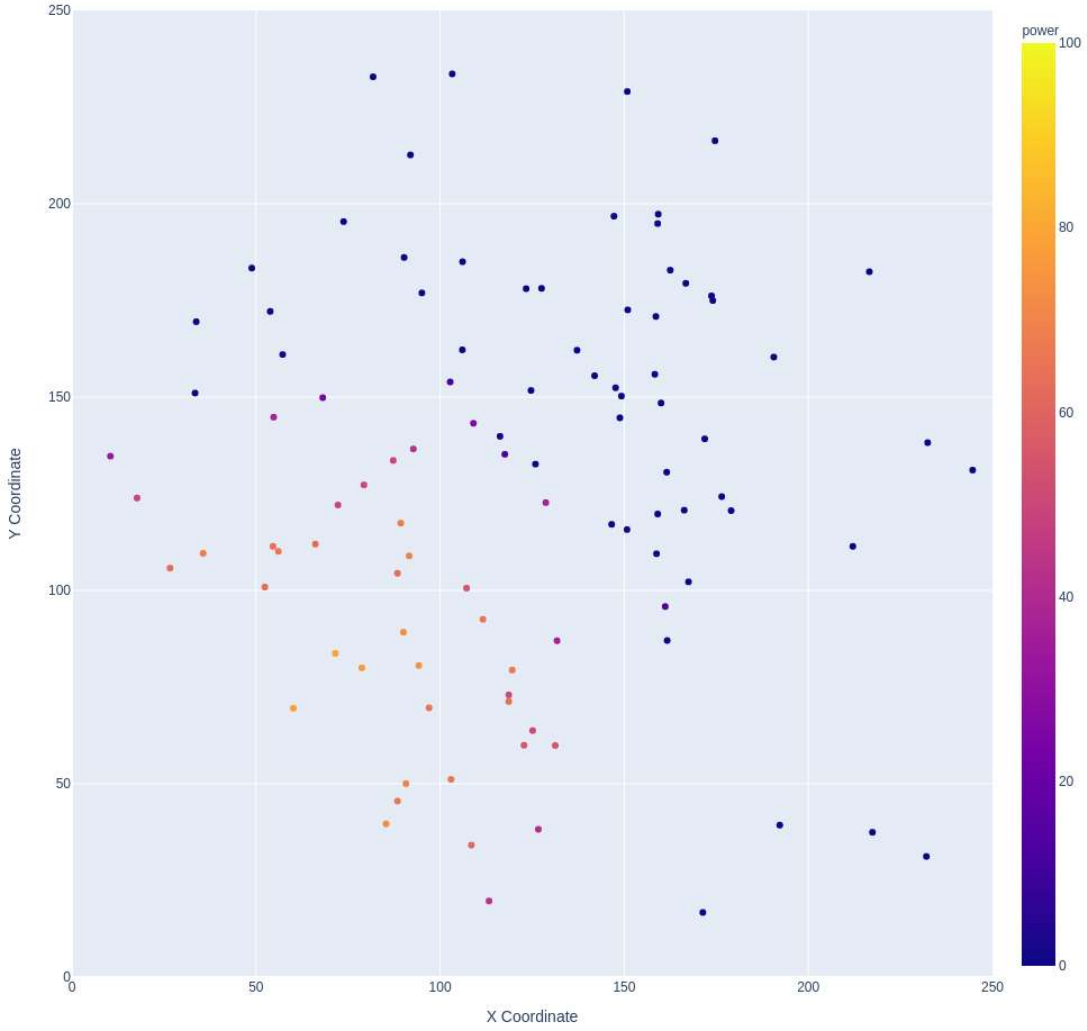


Figure 55: 100 Nodes 1 PPS LS-AODV Simulation

Table 7 shows the number of nodes remaining at the end of simulation that are still reachable. All twelve simulations show LS-AODV completing with more nodes still reachable than compared to EA-AODV.

# Nodes and Injection Rate	EA-AODV	LS-AODV	Number More Nodes Reachable in LS-AODV
50, 1pps	45	47	2
50, 2pps	36	43	7
50, 5pps	26	28	2
50, 10pps	18	22	4
75, 1pps	61	69	8
75, 2pps	51	58	7
75, 5pps	32	37	5
75, 10pps	24	31	7
100, 1pps	38	47	9
100, 2pps	69	73	4
100, 5pps	44	54	10
100, 10pps	39	51	12

Table 7: Number of Remaining Nodes in Networks.

Simulations show clear improvement in LS-AODV over EA-AODV as the network becomes larger and more congested. Network lifetime is used as a criteria to measure distribution of energy consumption more evenly through a network. The results of the simulations show that more nodes remained reachable in LS-AODV when compared to EA-AODV, and more reachable nodes means greater longevity of the battery-operated network.

Appendix A

SM14 and Bulldog Mote Firmware

ALS Sensor Firmware

als-sensor.h

```
/*
 * Copyright (c) 2013, ADVANSEE - http://www.advansee.com/
 * Benot Thbaudeau <benoit.thebaudeau@advansee.com>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * \addtogroup cc2538-bulldogV1-sensors
 * @{
 *
 * \defgroup bulldogV1-als-sensor BulldogV1 ALS Driver
 *
 * Driver for the BulldogV1 ALS sensor
 * @{
 *
 * \file
 * Header file for the bulldogV1 ALS Driver. Simple driver to tell ambient
 * lighting in a room. Converts ADC value for photoresistor into value from 0-100
 * where 0 represents a dark room with poor light, and 100 represents brightly
 * lit room.
 * Eqn:
 * ALS Val = ADC * (-1/160) + 106.25
 */
#ifndef ALS_SENSOR_H_
#define ALS_SENSOR_H_

#include "lib/sensors.h"

/*---------------------------------------------------------------------------*/
/** \name ALS sensor
```

```
 * @{
 */
#define ALS_SENSOR "ALS"
/** @} */

extern const struct sensors_sensor als_sensor;

#endif /* ALS_SENSOR_H_ */

/**
 * @}
 * @}
 */
```

als-sensor.c

```
/*
 * Copyright (c) 2013, ADVANSEE - http://www.advansee.com/
 * Benot Thbaudeau <benoit.thebaudeau@advansee.com>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * \addtogroup bulldogV1 ALS sensor
 * @{
 *
 * \file
 * Driver for the BulldobV1 ALS
 */
#include "contiki.h"
#include "sys/clock.h"
#include "dev/ioc.h"
#include "dev/gpio.h"
#include "dev/adc.h"
#include "dev/als-sensor.h"

#include <stdint.h>

#define ADC_ALS_PWR_PORT_BASE GPIO_PORT_TO_BASE(ADC_5V_PWR_EN_PORT)
#define ADC_ALS_PWR_PORT    ADC_5V_PWR_EN_PORT
#define ADC_ALS_PWR_PIN_MASK GPIO_PIN_MASK(ADC_5V_PWR_EN_PIN)
```

```
#define ADC_ALS_PWR_PIN     ADC_5V_PWR_EN_PIN
#define ADC_ALS_OUT_PIN_MASK GPIO_PIN_MASK(ADC_ALS_OUT_PIN)
// Analog sensors need many samples of readings, choose numbers of samples
// to be a power of 2 to use shifting instead of division for averaging
#define ADC_SAMPLES        128
#define ADC_LOG_2_SAMPLES     7
/*---------------------------------------------------------------------------*/
static int
value(int type)
{
  uint8_t channel = SOC_ADC_ADCCON_CH_AIN0 + ADC_ALS_OUT_PIN;
  int16_t res=0;
  int32_t res2 = 0;
  uint8_t i = 0;
  while (i < ADC_SAMPLES) {
    res = adc_get(channel, SOC_ADC_ADCCON_REF_AVDD5, SOC_ADC_ADCCON_DIV_512);
    res2 += (res & 0x2000) ? (res | 0xFFFFF000) : res;
    i++;
  }
  res2 = res2 >> ADC_LOG_2_SAMPLES;
  //res = adc_get(channel, SOC_ADC_ADCCON_REF_AVDD5, SOC_ADC_ADCCON_DIV_512);
  // need sign extension
  return res2;
}
/*---------------------------------------------------------------------------*/
static int
configure(int type, int value)
{
  switch(type) {
  case SENSORS_HW_INIT:
    GPIO_SOFTWARE_CONTROL(ADC_ALS_PWR_PORT_BASE, ADC_ALS_PWR_PIN_MASK);
    GPIO_SET_OUTPUT(ADC_ALS_PWR_PORT_BASE, ADC_ALS_PWR_PIN_MASK);
    GPIO_CLR_PIN(ADC_ALS_PWR_PORT_BASE, ADC_ALS_PWR_PIN_MASK);
    ioc_set_over(ADC_ALS_PWR_PORT, ADC_ALS_PWR_PIN, IOC_OVERRIDE_DIS);

    GPIO_SOFTWARE_CONTROL(GPIO_A_BASE, ADC_ALS_OUT_PIN_MASK);
    GPIO_SET_INPUT(GPIO_A_BASE, ADC_ALS_OUT_PIN_MASK);
    ioc_set_over(GPIO_A_NUM, ADC_ALS_OUT_PIN, IOC_OVERRIDE_ANA);

    break;
  case SENSORS_ACTIVE:
    if (value == 1) {
      // turn on PWR pin for ADC
      GPIO_SET_PIN(ADC_ALS_PWR_PORT_BASE, ADC_ALS_PWR_PIN_MASK);
    } else if (value == 0) {
      // turn off PWR pin for ADC
      GPIO_CLR_PIN(ADC_ALS_PWR_PORT_BASE, ADC_ALS_PWR_PIN_MASK);
    }
    break;

  }
  return 0;
}
/*---------------------------------------------------------------------------*/
static int
status(int type)
{
  return 1;
}
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR(als_sensor, ALS_SENSOR, value, configure, status);

/** @} */
```

BMP280 Sensor Firmware

bmp280.h

```
#define BMP280_ADDR 0x76

#define BMP280_REG_CHIP_ID 0xD0
#define BMP280_REG_RESET 0xE0
#define BMP280_REG_STATUS 0xF3
#define BMP280_REG_CONTROL 0xF4
#define BMP280_REG_CONFIG 0xF5
#define BMP280_REG_DATA     0xF7
#define BMP280_CHIP_ID 0x58
#define BMP280_REG_T1 0x88
#define BMP280_REG_T2 0x8A
#define BMP280_REG_T3 0x8C
#define BMP280_REG_P1 0x8E
#define BMP280_REG_P2 0x90
#define BMP280_REG_P3 0x92
#define BMP280_REG_P4 0x94
#define BMP280_REG_P5 0x96
#define BMP280_REG_P6 0x98
#define BMP280_REG_P7 0x9A
#define BMP280_REG_P8 0x9C
#define BMP280_REG_P9 0x9E
#define BMP280_RESET_VAL 0xB6

enum {
  BMP280_TEMP_READ,
  BMP280_PRESS_READ
};

typedef struct BMP280_CALIBRATION_DATA {
  uint16_t dig_T1;
  int16_t dig_T2;
  int16_t dig_T3;
  uint16_t dig_P1;
  int16_t dig_P2;
  int16_t dig_P3;
  int16_t dig_P4;
  int16_t dig_P5;
  int16_t dig_P6;
  int16_t dig_P7;
  int16_t dig_P8;
  int16_t dig_P9;
} BMP280_CALIBRATION_DATA;

uint8_t bmp280_init(uint8_t mode);
uint8_t bmp280_read(uint8_t type, int32_t *temperature, uint32_t *pressure);
```

bmp280.c

```
#include "contiki.h"
#include "sys/timer.h"
#include "bmp280.h"
#include "dev/board-i2c.h"

#define BMP280_STANDBY_TIME 250   // in microseconds

BMP280_CALIBRATION_DATA calib;

int bmp280_check_id() {
  uint8_t chipID;
  board_i2c_read(BMP280_ADDR,BMP280_REG_CHIP_ID,&chipID,1);
```

```
13   if (chipID == BMP280_CHIP_ID) return 0;
14   else return 1;
15 }
16
17 int bmp280_reset_chip() {
18   return board_i2c_write(BMP280_ADDR,BMP280_REG_RESET,BMP280_RESET_VAL);
19 }
20
21 int bmp280_read_status(uint8_t *status) {
22   return board_i2c_read(BMP280_ADDR,BMP280_REG_STATUS,status,1);
23 }
24
25 int bmp280_read_calib_data() {
26   if (
27     board_i2c_read16(BMP280_ADDR,BMP280_REG_T1, &calib.dig_T1) ||
28     board_i2c_read16(BMP280_ADDR,BMP280_REG_T2,(uint16_t *) &calib.dig_T2) ||
29     board_i2c_read16(BMP280_ADDR,BMP280_REG_T3,(uint16_t *) &calib.dig_T3) ||
30     board_i2c_read16(BMP280_ADDR,BMP280_REG_P1, &calib.dig_P1) ||
31     board_i2c_read16(BMP280_ADDR,BMP280_REG_P2,(uint16_t *) &calib.dig_P2) ||
32     board_i2c_read16(BMP280_ADDR,BMP280_REG_P3,(uint16_t *) &calib.dig_P3) ||
33     board_i2c_read16(BMP280_ADDR,BMP280_REG_P4,(uint16_t *) &calib.dig_P4) ||
34     board_i2c_read16(BMP280_ADDR,BMP280_REG_P5,(uint16_t *) &calib.dig_P5) ||
35     board_i2c_read16(BMP280_ADDR,BMP280_REG_P6,(uint16_t *) &calib.dig_P6) ||
36     board_i2c_read16(BMP280_ADDR,BMP280_REG_P7,(uint16_t *) &calib.dig_P7) ||
37     board_i2c_read16(BMP280_ADDR,BMP280_REG_P8,(uint16_t *) &calib.dig_P8) ||
38     board_i2c_read16(BMP280_ADDR,BMP280_REG_P9,(uint16_t *) &calib.dig_P9)
39   ) return 1;
40   else return 0;
41 }
42
43 int bmp280_configuration() {
44   uint8_t config = (3<<5); // 250 ms standby
45   config |= (0<<2); // IIR filter off
46   config |= 0; // disable SPI, want I2C instead
47   return board_i2c_write(BMP280_ADDR,BMP280_REG_CONFIG,config);
48 }
49
50 int bmp280_control() {
51   uint8_t ctrl = 3; // b11 is normal mode
52   ctrl |= (3<<5); // temp oversampleing of 4x
53   ctrl |= (3<<2); // pressure oversampling of 4x
54   return board_i2c_write(BMP280_ADDR,BMP280_REG_CONTROL,ctrl);
55 }
56
57 uint8_t bmp280_init(uint8_t mode) {
58   if (board_i2c_init() || bmp280_check_id()
59     || bmp280_reset_chip()) return 1;
60
61   // wait for power-on-reset and NVM data copy
62   while (1) {
63     uint8_t statusReg;
64     if (!bmp280_read_status(&statusReg) && ((statusReg & 1) == 0))
65     break;
66   }
67   if (
68     bmp280_read_calib_data() ||
69     bmp280_configuration() ||
70     bmp280_control()
71   ) return 1;
72
73   // wait for standby
74   for (int i = 0; i < 1000; i++) {
75     clock_delay_usec(BMP280_STANDBY_TIME);
76   }
77   return 0;
78 }
79
80 int bmp280_get_adc_vals(int32_t *adc_press,int32_t *adc_temp) {
```

```
  uint8_t data[6];
  int val = board_i2c_read(BMP280_ADDR,BMP280_REG_DATA,data,6);
  *(adc_press) = data[0] << 12 | data[1] << 4 | data[2] >> 4;
  *(adc_temp) = data[3] << 12 | data[4] << 4 | data[5] >> 4;
  return val;
}

static void bmp280_compensate_temp(int32_t *fix_temp, int32_t *fine_temp, int32_t *adcTemp) {
  int32_t v1, v2;
  int32_t adc_temp = *(adcTemp);

  v1 = ((((adc_temp>>3)-((int32_t)calib.dig_T1<<1)))*((int32_t)calib.dig_T2))>>11;
  v2 = (((((adc_temp>>4)-((int32_t)calib.dig_T1))*((adc_temp>>4)-((int32_t)calib.dig_T1)))\
    >>12)*((int32_t)calib.dig_T3))>>14;
  if (fine_temp != NULL) *(fine_temp) = v1+v2;
  *(fix_temp) = ((v1+v2)*5+128) >> 8;
}

static void bmp280_compensate_vals(int32_t *fix_temp, uint32_t *fix_press,
  int32_t *adc_temp, int32_t *adc_press) {

  int32_t fine_temp,v1,v2;
  bmp280_compensate_temp(fix_temp,&fine_temp,adc_temp);
  v1 = (fine_temp >> 1) - (int32_t)64000;
  v2 = (((v1>>2)*(v1>>2))>>11)*((int32_t)calib.dig_P6);
  v2 = v2 + ((v1*((int32_t)calib.dig_P5))<<1);
  v2 = (v2>>2)+(((int32_t)calib.dig_P4)<<16);
  v1 = (((calib.dig_P3*(((v1>>2)*(v1>>2))>>13))>>3)+((((int32_t)calib.dig_P2)*
    v1)>>1))>>18;
  v1 = ((((32768+v1))*((int32_t)calib.dig_P1))>>15);
  if (v1 == 0) {
    *(fix_press) = 0;
    return;
  }
  uint32_t p = (((uint32_t)(((int32_t)1048576)-*(adc_press))-(v2>>12)))*3125;
  if (p < 0x80000000) p = (p << 1) / ((uint32_t)v1);
  else p = (p / (uint32_t)v1)*2;
  v1 = (((int32_t)calib.dig_P9)*((int32_t)(((p>>3)*(p>>3))>>13)))>>12;
  v2 = (((int32_t)(p>>2))*((int32_t)calib.dig_P8))>>13;
  p = (uint32_t)((int32_t)p+((v1+v2+calib.dig_P7)>>4));
  *(fix_press) = p;
  return;
}

uint8_t bmp280_read(uint8_t type, int32_t *temp, uint32_t *pressure) {
  int32_t fix_temp,adc_press,adc_temp;
  uint32_t fix_press;

  //printf("%d %d %d\n%d %d %d %d %d %d %d %d\n",calib.dig_T1,calib.dig_T2,
  // calib.dig_T3,calib.dig_P1,calib.dig_P2,calib.dig_P3,calib.dig_P4,calib.dig_P5,
  // calib.dig_P6,calib.dig_P7,calib.dig_P8,calib.dig_P9);

  if (bmp280_get_adc_vals(&adc_press,&adc_temp)) return 1;
  if (type == BMP280_TEMP_READ) {
    bmp280_compensate_temp(&fix_temp,NULL,&adc_temp);
    *temp = fix_temp;
  } else {
    bmp280_compensate_vals(&fix_temp,&fix_press,&adc_temp,&adc_press);
    *pressure = fix_press;
  }
  return 0;
}
```

bmp280-sensor.h

```
/*
```

```
 * Copyright (c) 2021, Copyright Calvin Jarrod Smith
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ‘‘AS IS’’ AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * This file is used in the Contiki software for the BMP280 temperature and
 * pressure sensor.
 *
 *
 * Author : Calvin Jarrod Smith, calvin.jarrod@gmail.com
 * Created : 2021-02-16
 */

#ifndef BMP280_SENSOR_H_
#define BMP280_SENSOR_H_

#include "lib/sensors.h"

extern const struct sensors_sensor bmp280_sensor;

#endif /* BMP280_SENSOR_H_ */
```

bmp280-sensor.c

```
#include "contiki.h"
#include "lib/sensors.h"
#include "dev/bmp280.h"
#include "dev/bmp280-sensor.h"
#include "dev/board-i2c.h"

static int
value(int type) {
  int32_t temp;
  uint32_t press;

  bmp280_read(type,&temp,&press);
  if (type == BMP280_TEMP_READ) return (int)temp;
  else if (type == BMP280_PRESS_READ) return (int)press;
  else return 0;
}

static int
status(int type) {
  return 0;
```

```
}

static int
configure(int type, int c) {
  board_i2c_init();
  return bmp280_init(type);
}

SENSORS_SENSOR(bmp280_sensor,"BMP280",value,configure,status);
```

Board Pinout Firmware

board.h

```
/*
 * Copyright (c) 2012, Texas Instruments Incorporated - http://www.ti.com/
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * \addtogroup bulldogV1
 * @{
 *
 * \defgroup Bulldog Mote V1 and SM14Z2538 Board Peripherals
 *
 * Defines related to the Bulldog Mote V1 and SM14Z2538 Boards
 *
 * This file provides connectivity information on LEDs, Buttons, UART
 *
 * This file can be used as the basis to configure other platforms using the
 * cc2538 SoC.
 * @{
 *
 * \file
 * Header file with definitions related to the I/O connections on the
 * Bulldog Mote V1 and SM14Z2538 Development boards
 *
 * \note Do not include this file directly. It gets included by contiki-conf
 * after all relevant directives have been set.
```

```
 */
#ifndef BOARD_H_
#define BOARD_H_

#include "dev/gpio.h"
#include "dev/nvic.h"
/*---------------------------------------------------------------------------*/
/** \name SmartRF LED configuration
 *
 * LEDs on the SM14Z2538 and BulldogV1 (EB and BB) are connected as follows:
 * - LED4 (Red) -> PC4
 * - LED3 (Yellow) -> PC5
 * - LED2 (Green) -> PC6
 * - LED1 (Blue) -> PC7
 *
 * LED1 shares the same pin with the USB pullup
 * @{
 */
/*---------------------------------------------------------------------------*/

#define LEDS_CONF_RED 1
#define LEDS_CONF_YELLOW 2
#define LEDS_CONF_GREEN 4
#define LEDS_CONF_BLUE 8
#define LEDS_CONF_ALL (LEDS_CONF_RED|LEDS_CONF_YELLOW|LEDS_CONF_GREEN|LEDS_CONF_BLUE)

#define LEDS_ARCH_L1_PORT  GPIO_C_NUM
#define LEDS_ARCH_L1_PIN  4
#define LEDS_ARCH_L2_PORT  GPIO_C_NUM
#define LEDS_ARCH_L2_PIN   5
#define LEDS_ARCH_L3_PORT  GPIO_C_NUM
#define LEDS_ARCH_L3_PIN   6
#define LEDS_ARCH_L4_PORT  GPIO_C_NUM
#define LEDS_ARCH_L4_PIN  7

#define LEDS_CONF_COUNT 4
/** @} */
/*---------------------------------------------------------------------------*/
/** \name USB configuration
 *
 * The USB pullup is driven by PC0 and is shared with LED1
 */
#define USB_PULLUP_PORT GPIO_C_NUM
#define USB_PULLUP_PIN 0
/** @} */
/*---------------------------------------------------------------------------*/
/** \name UART configuration
 *
 * - RX: PA0
 * - TX: PA1
 * - CTS: PB0 (Can only be used with UART1)
 * - RTS: PD3 (Can only be used with UART1)
 *
 * We configure the port to use UART0. To use UART1, replace UART0_* with
 * UART1_* below.
 * @{
 */
#define UART0_RX_PORT GPIO_A_NUM
#define UART0_RX_PIN 0

#define UART0_TX_PORT GPIO_A_NUM
#define UART0_TX_PIN 1

#define UART1_CTS_PORT GPIO_B_NUM
#define UART1_CTS_PIN 2

#define UART1_RTS_PORT GPIO_D_NUM
#define UART1_RTS_PIN 3
```

```
119 /** @} */
120 /*---------------------------------------------------------------------------*/
121 /** \name SM14 Button configuration
122  *
123  * Buttons on the SmartRF06 are connected as follows:
124  * - BUTTON_LEFT -> PC1
125  * - BUTTON_RIGHT -> PC0
126  * @{
127  */
128 /** BUTTON_LEFT -> PC1 */
129 #define BUTTON_LEFT_PORT GPIO_C_NUM
130 #define BUTTON_LEFT_PIN 1
131 #define BUTTON_LEFT_VECTOR GPIO_C_IRQn
132
133 /** BUTTON_RIGHT -> PC0 */
134 #define BUTTON_RIGHT_PORT GPIO_C_NUM
135 #define BUTTON_RIGHT_PIN 0
136 #define BUTTON_RIGHT_VECTOR GPIO_C_IRQn
137
138 /* Notify various examples that we have Buttons */
139 #define PLATFORM_HAS_BUTTON 1
140 #define PLATFORM_SUPPORTS_BUTTON_HAL 1
141 /** @} */
142 /*---------------------------------------------------------------------------*/
143 /**
144  * \name ADC configuration
145  *
146  * These values configure which CC2538 pins and ADC channels to use for the ADC
147  * inputs.
148  *
149  * ADC inputs can only be on port A.
150  * @{
151  */
152 //#define ADC_ALS_PWR_PORT GPIO_A_NUM /**< ALS power GPIO control port */
153 //#define ADC_ALS_PWR_PIN 7 /**< ALS power GPIO control pin */
154 #define ADC_ALS_OUT_PIN 6 /**< ALS output ADC input pin on port A */
155
156 //#define ADC_MQ135_PWR_PORT   GPIO_A_NUM
157 //#define ADC_MQ135_PWR_PIN 4 /**< GPIO connected to 5V regulator EN */
158 #define ADC_MQ135_OUT_PIN    5 /**< MQ135 output ADC input pin on port A */
159
160 #define ADC_5V_PWR_EN_PORT GPIO_A_NUM /**< ALS power GPIO control port */
161 #define ADC_5V_PWR_EN_PIN    4
162 /** @} */
163 /*---------------------------------------------------------------------------*/
164 /**
165  * \name SPI configuration
166  *
167  * These values configure which CC2538 pins to use for the SPI lines. Both
168  * SPI instances can be used independently by providing the corresponding
169  * port / pin macros.
170  * @{
171  */
172 #define SPI0_IN_USE 1
173 #define SPI1_IN_USE 0
174 #if SPI0_IN_USE
175 /** Clock port SPI0 */
176 #define SPI0_CLK_PORT GPIO_D_NUM
177 /** Clock pin SPI0 */
178 #define SPI0_CLK_PIN 0
179 /** TX port SPI0 (master mode: MOSI) */
180 #define SPI0_TX_PORT GPIO_D_NUM
181 /** TX pin SPI0 */
182 #define SPI0_TX_PIN 1
183 /** RX port SPI0 (master mode: MISO */
184 #define SPI0_RX_PORT GPIO_D_NUM
185 /** RX pin SPI0 */
186 #define SPI0_RX_PIN 3
```

```
187 #endif /* #if SPI0_IN_USE */
188 #if SPI1_IN_USE
189 /** Clock port SPI1 */
190 #define SPI1_CLK_PORT GPIO_D_NUM
191 /** Clock pin SPI1 */
192 #define SPI1_CLK_PIN 0
193 /** TX port SPI1 (master mode: MOSI) */
194 #define SPI1_TX_PORT GPIO_D_NUM
195 /** TX pin SPI1 */
196 #define SPI1_TX_PIN 1
197 /** RX port SPI1 (master mode: MISO) */
198 #define SPI1_RX_PORT GPIO_D_NUM
199 /** RX pin SPI1 */
200 #define SPI1_RX_PIN 3
201 #endif /* #if SPI1_IN_USE */
202 /** @} */
203 /*---------------------------------------------------------------------------*/
204 /**
205  * \name AT25SF321A Flash Memory Chip Select Line
206  *
207  * @{
208  */
209 #define FLASH_SPI_CS_PORT    GPIO_D_NUM
210 #define FLASH_SPI_CS_PIN    4
211
212 /** @} */
213 /*---------------------------------------------------------------------------*/
214 /**
215  * \name CC2538 TSCH configuration
216  *
217  * @{
218  */
219 #define RADIO_PHY_OVERHEAD CC2538_PHY_OVERHEAD
220 #define RADIO_BYTE_AIR_TIME CC2538_BYTE_AIR_TIME
221 #define RADIO_DELAY_BEFORE_TX CC2538_DELAY_BEFORE_TX
222 #define RADIO_DELAY_BEFORE_RX CC2538_DELAY_BEFORE_RX
223 #define RADIO_DELAY_BEFORE_DETECT CC2538_DELAY_BEFORE_DETECT
224 /** @} */
225 /*---------------------------------------------------------------------------*/
226 /**
227  * \name I2C configuration
228  *
229  * As default there is not a default pin assignment for I2C, change this values
230  * accordingly if mapping either pin to the I2C controller.
231  * @{
232  */
233 #define I2C_SCL_PORT GPIO_B_NUM
234 #define I2C_SCL_PIN 0
235 #define I2C_SDA_PORT GPIO_B_NUM
236 #define I2C_SDA_PIN 1
237 #define I2C_INT_PORT      GPIO_B_NUM
238 #define I2C_INT_PIN       3
239 /** @} */
240 /*---------------------------------------------------------------------------*/
241 /**
242  * \name CC2592 Range Entender
243  *
244  * @{
245  */
246 #define RE_PA_EN_PORT      GPIO_C_NUM
247 #define RE_PA_EN_PIN      3
248 #define RE_LNA_EN_PORT     GPIO_C_NUM
249 #define RE_LNA_EN_PIN      2
250 #define RE_HGM_PORT       GPIO_D_NUM
251 #define RE_HGM_PIN       2
252 /** @} */
253 /**
254  * \name Device string used on startup
```

```
 * @{
 */
#define BOARD_STRING "BULLDOG␣MOTE␣V1"
/** @} */

#endif /* BOARD_H_ */

/**
 * @}
 * @}
 */
```

I2C Firmware

board-i2c.h

```
/*
 * Copyright (c) 2016, Zolertia <http://www.zolertia.com>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
/*---------------------------------------------------------------------------*/
#include "contiki.h"

/* Initialize the I2C module */
uint8_t board_i2c_init();

/* I2C read registers */
uint8_t board_i2c_read(uint8_t addr, uint8_t reg, uint8_t *buf,
                       uint8_t bytes);

/* I2C write to a single register */
uint8_t board_i2c_write(uint8_t addr, uint8_t reg, uint8_t value);

uint8_t board_i2c_read16(uint8_t addr, uint8_t reg, uint16_t *buf);

/*---------------------------------------------------------------------------*/
/** @} */
```

board-i2c.c

```c
/*
 * Copyright (c) 2015, Zolertia
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/*---------------------------------------------------------------------------*/
/**
 * \addtogroup zoul-bme280-sensor
 * \ingroup zoul
 *
 * @{
 *
 * \file
 * Architecture-specific I2C for the external BME280 weather sensor
 *
 * \author
 * Antonio Lignan <alinan@zolertia.com>
 */
/*---------------------------------------------------------------------------*/
#include "contiki.h"
#include "dev/i2c.h"
/*---------------------------------------------------------------------------*/
uint8_t
board_i2c_init(void)
{
  i2c_init(I2C_SDA_PORT, I2C_SDA_PIN, I2C_SCL_PORT, I2C_SCL_PIN,
           I2C_SCL_FAST_BUS_SPEED);
  return i2c_master_error();
}
/*---------------------------------------------------------------------------*/
uint8_t
board_i2c_write(uint8_t addr, uint8_t reg, uint8_t value)
{
  uint8_t buf[2];

  buf[0] = reg;
  buf[1] = value;

  i2c_master_enable();
  return i2c_burst_send(addr, buf, 2);
}
```

```
/*---------------------------------------------------------------------------*/
uint8_t
board_i2c_read(uint8_t addr, uint8_t reg, uint8_t *buf, uint8_t bytes)
{
  i2c_master_enable();
  if(i2c_single_send(addr, reg) == I2C_MASTER_ERR_NONE) {
    while(i2c_master_busy());
    return i2c_burst_receive(addr, buf, bytes);
  } else return I2CM_STAT_ERROR;
}
/*---------------------------------------------------------------------------*/
uint8_t
board_i2c_read16(uint8_t addr, uint8_t reg, uint16_t *buf)
{
  uint8_t data[2];
  i2c_master_enable();
  if(i2c_single_send(addr, reg) == I2C_MASTER_ERR_NONE) {
    while(i2c_master_busy());
    int ret = i2c_burst_receive(addr, data, 2);
    *buf = (uint16_t)data[1] << 8 | data[0];
    return ret;
  } else return I2CM_STAT_ERROR;
}
/**
 * @}
 */
```

Board Sensor Firmware

bulldogV1-sensors.c

```
/*
 * Copyright (c) 2012, Texas Instruments Incorporated - http://www.ti.com/
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * \addtogroup cc2538-smartrf
 * @{
```

```
 *
 * \defgroup cc2538-smartrf-sensors SmartRF06EB Sensors
 *
 * Generic module controlling sensors on the SmartRF06EB
 * @{
 *
 * \file
 * Implementation of a generic module controlling SmartRF06EB sensors
 */
#include "contiki.h"
#include "lib/sensors.h"
#include "dev/als-sensor.h"
#include "dev/bmp280-sensor.h"
#include "dev/mq135-sensor.h"
#include "dev/cc2538-sensors.h"

#include <string.h>

/** \brief Exports a global symbol to be used by the sensor API */
SENSORS(&als_sensor, &cc2538_temp_sensor, &vdd3_sensor, &bmp280_sensor, &mq135_sensor);
//SENSORS(&als_sensor, &cc2538_temp_sensor, &vdd3_sensor, &bmp280_sensor);

/**
 * @}
 * @}
 */
```

MQ135 Sensor Firmware

mq135-sensor.h

```
/*
 * Copyright (c) 2013, ADVANSEE - http://www.advansee.com/
 * Benot Thbaudeau <benoit.thebaudeau@advansee.com>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * \addtogroup cc2538-bulldogV1-sensors
```

```
 * @{
 *
 * \defgroup bulldogV1-als-sensor BulldogV1 ALS Driver
 *
 * Driver for the BulldogV1 ALS sensor
 * @{
 *
 * \file
 * Header file for the bulldogV1 ALS Driver. Simple driver to tell ambient
 * lighting in a room. Converts ADC value for photoresistor into value from 0-100
 * where 0 represents a dark room with poor light, and 100 represents brightly
 * lit room.
 * Eqn:
 * ALS Val = ADC * (-1/160) + 106.25
 */
#ifndef MQ135_SENSOR_H_
#define MQ135_SENSOR_H_

#include "lib/sensors.h"

/*---------------------------------------------------------------------------*/
/** \name ALS sensor
 * @{
 */
#define MQ135_SENSOR "MQ135"
/** @} */

extern const struct sensors_sensor mq135_sensor;

#endif /* ALS_SENSOR_H_ */

/**
 * @}
 * @}
 */
```

mq135-sensor.c

```
/*
 * Copyright (c) 2013, ADVANSEE - http://www.advansee.com/
 * Benot Thbaudeau <benoit.thebaudeau@advansee.com>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
```

```
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * \addtogroup bulldogV1 MQ135 sensor
 * @{
 *
 * \file
 * Driver for the BulldobV1 MQ135
 */
#include "contiki.h"
#include "sys/clock.h"
#include "dev/ioc.h"
#include "dev/gpio.h"
#include "dev/adc.h"
#include "dev/mq135-sensor.h"

#include <stdint.h>

#define ADC_MQ135_PWR_PORT_BASE GPIO_PORT_TO_BASE(ADC_5V_PWR_EN_PORT)
#define ADC_MQ135_PWR_PORT    ADC_5V_PWR_EN_PORT
#define ADC_MQ135_PWR_PIN_MASK GPIO_PIN_MASK(ADC_5V_PWR_EN_PIN)
#define ADC_MQ135_PWR_PIN ADC_5V_PWR_EN_PIN
#define ADC_MQ135_OUT_PIN_MASK GPIO_PIN_MASK(ADC_MQ135_OUT_PIN)
// Analog sensors need many samples of readings, choose numbers of samples
// to be a power of 2 to use shifting instead of division for averaging
#define ADC_SAMPLES        128
#define ADC_LOG_2_SAMPLES     7
/*---------------------------------------------------------------------------*/
static int
value(int type)
{
  uint8_t channel = SOC_ADC_ADCCON_CH_AIN0 + ADC_MQ135_OUT_PIN;
  int16_t res = 0;
  uint8_t i = 0;
  while (i < ADC_SAMPLES) {
    res += adc_get(channel, SOC_ADC_ADCCON_REF_AVDD5, SOC_ADC_ADCCON_DIV_512);
    i++;
  }
  uint16_t res2 = res >> ADC_LOG_2_SAMPLES;
  return res2;
}
/*---------------------------------------------------------------------------*/
static int
configure(int type, int value)
{
  switch(type) {
  case SENSORS_HW_INIT:
    // configure PWR pin
    GPIO_SOFTWARE_CONTROL(ADC_MQ135_PWR_PORT_BASE, ADC_MQ135_PWR_PIN_MASK);
    GPIO_SET_OUTPUT(ADC_MQ135_PWR_PORT_BASE, ADC_MQ135_PWR_PIN_MASK);
    GPIO_CLR_PIN(ADC_MQ135_PWR_PORT_BASE, ADC_MQ135_PWR_PIN_MASK);
    ioc_set_over(ADC_MQ135_PWR_PORT, ADC_MQ135_PWR_PIN, IOC_OVERRIDE_DIS);

    // configure ADC input pin
    GPIO_SOFTWARE_CONTROL(GPIO_A_BASE, ADC_MQ135_OUT_PIN_MASK);
    GPIO_SET_INPUT(GPIO_A_BASE, ADC_MQ135_OUT_PIN_MASK);
    ioc_set_over(GPIO_A_NUM, ADC_MQ135_OUT_PIN, IOC_OVERRIDE_ANA);

    break;
  case SENSORS_ACTIVE:
    if (value == 1) {
      // turn on PWR pin for ADC
      GPIO_SET_PIN(ADC_MQ135_PWR_PORT_BASE, ADC_MQ135_PWR_PIN_MASK);
    } else if (value == 0) {
      // turn off PWR pin for ADC
      GPIO_CLR_PIN(ADC_MQ135_PWR_PORT_BASE, ADC_MQ135_PWR_PIN_MASK);
    }
```

```
    break;
  }

  return 0;
}
/*---------------------------------------------------------------------------*/
static int
status(int type)
{
  return 1;
}
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR(mq135_sensor, MQ135_SENSOR, value, configure, status);

/** @} */
```

Platform Firmware

platform.c

```
/*
 * Copyright (c) 2012, Texas Instruments Incorporated - http://www.ti.com/
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * \addtogroup cc2538-platforms
 * @{
 *
 * \defgroup cc2538dk The cc2538 Development Kit platform
 *
 * The cc2538DK is a platform by Texas Instruments, based on the
 * cc2538 SoC with an ARM Cortex-M3 core.
 * @{
 *
 * \file
 * Main module for the bulldogV1 platform
 */
/*---------------------------------------------------------------------------*/
```

```
#include "contiki.h"
#include "dev/adc.h"
#include "dev/leds.h"
#include "dev/uart.h"
#include "dev/serial-line.h"
#include "dev/slip.h"
#include "dev/cc2538-rf.h"
#include "dev/udma.h"
#include "dev/crypto.h"
#include "dev/button-hal.h"
#include "usb/usb-serial.h"
#include "lib/random.h"
#include "lib/sensors.h"
#include "net/netstack.h"
#include "net/mac/framer/frame802154.h"
#include "net/linkaddr.h"
#include "sys/platform.h"
#include "soc.h"
#include "cpu.h"
#include "reg.h"
#include "ieee-addr.h"
#include "lpm.h"

#include <stdint.h>
#include <string.h>
#include <stdio.h>
/*---------------------------------------------------------------------------*/
/* Log configuration */
#include "sys/log.h"
#define LOG_MODULE "BULLDOGV1"
#define LOG_LEVEL LOG_LEVEL_MAIN
/*---------------------------------------------------------------------------*/
static void
fade(leds_mask_t l)
{
  volatile int i;
  int k, j;
  for(k = 0; k < 800; ++k) {
    j = k > 400 ? 800 - k : k;

    leds_on(l);
    for(i = 0; i < j; ++i) {
      __asm("nop");
    }
    leds_off(l);
    for(i = 0; i < 400 - j; ++i) {
      __asm("nop");
    }
  }
}
/*---------------------------------------------------------------------------*/
static void
set_rf_params(void)
{
  uint16_t short_addr;
  uint8_t ext_addr[8];

  ieee_addr_cpy_to(ext_addr, 8);

  short_addr = ext_addr[7];
  short_addr |= ext_addr[6] << 8;

  NETSTACK_RADIO.set_value(RADIO_PARAM_PAN_ID, IEEE802154_PANID);
  NETSTACK_RADIO.set_value(RADIO_PARAM_16BIT_ADDR, short_addr);
  NETSTACK_RADIO.set_value(RADIO_PARAM_CHANNEL, IEEE802154_DEFAULT_CHANNEL);
  NETSTACK_RADIO.set_object(RADIO_PARAM_64BIT_ADDR, ext_addr, 8);
}
/*---------------------------------------------------------------------------*/
```

```
113 void
114 platform_init_stage_one(void)
115 {
116   soc_init();
117
118   leds_init();
119   fade(LEDS_RED);
120 }
121 /*---------------------------------------------------------------------------*/
122 void
123 platform_init_stage_two()
124 {
125   /*
126    * Character I/O Initialisation.
127    * When the UART receives a character it will call serial_line_input_byte to
128    * notify the core. The same applies for the USB driver.
129    *
130    * If slip-arch is also linked in afterwards (e.g. if we are a border router)
131    * it will overwrite one of the two peripheral input callbacks. Characters
132    * received over the relevant peripheral will be handled by
133    * slip_input_byte instead
134    */
135 #if UART_CONF_ENABLE
136   uart_init(0);
137   uart_init(1);
138   uart_set_input(SERIAL_LINE_CONF_UART, serial_line_input_byte);
139 #endif
140
141 #if USB_SERIAL_CONF_ENABLE
142   usb_serial_init();
143   usb_serial_set_input(serial_line_input_byte);
144 #endif
145
146   serial_line_init();
147
148   /* Initialise the H/W RNG engine. */
149   random_init(0);
150
151   udma_init();
152
153 #if CRYPTO_CONF_INIT
154   crypto_init();
155   crypto_disable();
156 #endif
157
158   /* Populate linkaddr_node_addr */
159   ieee_addr_cpy_to(linkaddr_node_addr.u8, LINKADDR_SIZE);
160
161   button_hal_init();
162
163   INTERRUPTS_ENABLE();
164
165   fade(LEDS_YELLOW);
166 }
167 /*---------------------------------------------------------------------------*/
168 void
169 platform_init_stage_three()
170 {
171   LOG_INFO("%s\n", BOARD_STRING);
172
173   set_rf_params();
174
175   soc_print_info();
176
177   adc_init();
178
179   // set CC2592 pins as outputs
180   gpio_hal_arch_pin_set_output(CC2538_RF_RE_PA_PORT,CC2538_RF_RE_PA_PIN);
```

```
  gpio_hal_arch_pin_set_output(CC2538_RF_RE_LNA_PORT,CC2538_RF_RE_LNA_PIN);
  gpio_hal_arch_pin_set_output(CC2538_RF_RE_HGM_PORT,CC2538_RF_RE_HGM_PIN);
  // PA = 0, LNA = 1 for RX
  gpio_hal_arch_clear_pin(CC2538_RF_RE_PA_PORT,CC2538_RF_RE_PA_PIN);
  gpio_hal_arch_set_pin(CC2538_RF_RE_LNA_PORT,CC2538_RF_RE_LNA_PIN);
  #if CC2538_RF_RE_HGM_EN
  gpio_hal_arch_set_pin(CC2538_RF_RE_HGM_PORT,CC2538_RF_RE_HGM_PIN);
  #else
  gpio_hal_arch_clear_pin(CC2538_RF_RE_HGM_PORT,CC2538_RF_RE_HGM_PIN);
  #endif

  fade(LEDS_GREEN);
  process_start(&sensors_process, NULL);

  fade(LEDS_BLUE);
}
/*---------------------------------------------------------------------------*/
void
platform_idle()
{
  /* We have serviced all pending events. Enter a Low-Power mode. */
  lpm_enter();
}
/*---------------------------------------------------------------------------*/
/**
 * @}
 * @}
 */
```

Appendix B

LS-AODV Simulated Routing Protocols

EA-AODV

ea-aodv-routing-protocol.h

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * 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
 *
 * Based on
 * NS-2 AODV model developed by the CMU/MONARCH group and optimized and
 * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
 *
 * AODV-UU implementation by Erik Nordstrm of Uppsala University
 * http://core.it.uu.se/core/index.php/AODV-UU
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */
#ifndef EA_AODVROUTINGPROTOCOL_H
#define EA_AODVROUTINGPROTOCOL_H

#define ESP32_VOLTAGE 3.3f

// Cameron: This should change in the future when we allow variablity
#define PACKET_SIZE 4096
#define RREF_PACKET_SIZE 64

// Cameron: Burn rate for transmission per byte. Update this as determined.

#include "ea-aodv-rtable.h"
#include "ea-aodv-rqueue.h"
#include "ea-aodv-packet.h"
#include "ea-aodv-neighbor.h"
#include "ea-aodv-dpd.h"
#include "ns3/node.h"
#include "ns3/random-variable-stream.h"
#include "ns3/output-stream-wrapper.h"
#include "ns3/ipv4-routing-protocol.h"
#include "ns3/ipv4-interface.h"
#include "ns3/ipv4-l3-protocol.h"
#include <map>

namespace ns3 {
namespace eaAodv {
/**
 * \ingroup eaAodv
```

```
 *
 * \brief AODV routing protocol
 */
class RoutingProtocol : public Ipv4RoutingProtocol
{
public:
  /**
   * \brief Get the type ID.
   * \return the object TypeId
   */
  static TypeId GetTypeId (void);
  static const uint32_t EA_AODV_PORT;

  // EA-AODV
  static const float power_receive;
  static const float power_active;
  static const float m_aliveThreshold;
  static const double TxConst;

  /// constructor
  RoutingProtocol ();
  virtual ~RoutingProtocol ();
  virtual void DoDispose ();

  // Inherited from Ipv4RoutingProtocol
  Ptr<Ipv4Route> RouteOutput (Ptr<Packet> p, const Ipv4Header &header, Ptr<NetDevice> oif,
       ↪ Socket::SocketErrno &sockerr);
  bool RouteInput (Ptr<const Packet> p, const Ipv4Header &header, Ptr<const NetDevice> idev,
                   UnicastForwardCallback ucb, MulticastForwardCallback mcb,
                   LocalDeliverCallback lcb, ErrorCallback ecb);
  virtual void NotifyInterfaceUp (uint32_t interface);
  virtual void NotifyInterfaceDown (uint32_t interface);
  virtual void NotifyAddAddress (uint32_t interface, Ipv4InterfaceAddress address);
  virtual void NotifyRemoveAddress (uint32_t interface, Ipv4InterfaceAddress address);
  virtual void SetIpv4 (Ptr<Ipv4> ipv4);
  virtual void PrintRoutingTable (Ptr<OutputStreamWrapper> stream, Time::Unit unit = Time::S) const;

  // Handle protocol parameters
  /**
   * Get maximum queue time
   * \returns the maximum queue time
   */
  Time GetMaxQueueTime () const
  {
    return m_maxQueueTime;
  }
  /**
   * Set the maximum queue time
   * \param t the maximum queue time
   */
  void SetMaxQueueTime (Time t);
  /**
   * Get the maximum queue length
   * \returns the maximum queue length
   */
  uint32_t GetMaxQueueLen () const
  {
    return m_maxQueueLen;
  }
  /**
   * Set the maximum queue length
   * \param len the maximum queue length
   */
  void SetMaxQueueLen (uint32_t len);
  /**
   * Get destination only flag
   * \returns the destination only flag
   */
```

```
123   bool GetDestinationOnlyFlag () const
124   {
125     return m_destinationOnly;
126   }
127   /**
128    * Set destination only flag
129    * \param f the destination only flag
130    */
131   void SetDestinationOnlyFlag (bool f)
132   {
133     m_destinationOnly = f;
134   }
135   /**
136    * Get gratuitous reply flag
137    * \returns the gratuitous reply flag
138    */
139   bool GetGratuitousReplyFlag () const
140   {
141     return m_gratuitousReply;
142   }
143   /**
144    * Set gratuitous reply flag
145    * \param f the gratuitous reply flag
146    */
147   void SetGratuitousReplyFlag (bool f)
148   {
149     m_gratuitousReply = f;
150   }
151   /**
152    * Set hello enable
153    * \param f the hello enable flag
154    */
155   void SetHelloEnable (bool f)
156   {
157     m_enableHello = f;
158   }
159   /**
160    * Get hello enable flag
161    * \returns the enable hello flag
162    */
163   bool GetHelloEnable () const
164   {
165     return m_enableHello;
166   }
167   /**
168    * Set broadcast enable flag
169    * \param f enable broadcast flag
170    */
171   void SetBroadcastEnable (bool f)
172   {
173     m_enableBroadcast = f;
174   }
175   /**
176    * Get broadcast enable flag
177    * \returns the broadcast enable flag
178    */
179   bool GetBroadcastEnable () const
180   {
181     return m_enableBroadcast;
182   }
183
184   /**
185    * Assign a fixed random variable stream number to the random variables
186    * used by this model. Return the number of streams (possibly zero) that
187    * have been assigned.
188    *
189    * \param stream first stream index to use
190    * \return the number of stream indices assigned by this model
```

```
   */
  int64_t AssignStreams (int64_t stream);

protected:
  virtual void DoInitialize (void);
private:
  // Protocol parameters.
  uint32_t m_rreqRetries; ///< Maximum number of retransmissions of RREQ with TTL = NetDiameter to discover a
      ↪ route
  uint16_t m_ttlStart; ///< Initial TTL value for RREQ.
  uint16_t m_ttlIncrement; ///< TTL increment for each attempt using the expanding ring search for RREQ
      ↪ dissemination.
  uint16_t m_ttlThreshold; ///< Maximum TTL value for expanding ring search, TTL = NetDiameter is used beyond
      ↪ this value.
  uint16_t m_timeoutBuffer; ///< Provide a buffer for the timeout.
  uint16_t m_rreqRateLimit; ///< Maximum number of RREQ per second.
  uint16_t m_rerrRateLimit; ///< Maximum number of REER per second.
  Time m_activeRouteTimeout; ///< Period of time during which the route is considered to be valid.
  uint32_t m_netDiameter; ///< Net diameter measures the maximum possible number of hops between two nodes in
      ↪ the network
  /**
   * NodeTraversalTime is a conservative estimate of the average one hop traversal time for packets
   * and should include queuing delays, interrupt processing times and transfer times.
   */
  Time m_nodeTraversalTime;
  Time m_netTraversalTime; ///< Estimate of the average net traversal time.
  Time m_pathDiscoveryTime; ///< Estimate of maximum time needed to find route in network.
  Time m_myRouteTimeout; ///< Value of lifetime field in RREP generating by this node.
  /**
   * Every HelloInterval the node checks whether it has sent a broadcast within the last HelloInterval.
   * If it has not, it MAY broadcast a Hello message
   */
  Time m_helloInterval;
  uint32_t m_allowedHelloLoss; ///< Number of hello messages which may be loss for valid link
  /**
   * DeletePeriod is intended to provide an upper bound on the time for which an upstream node A
   * can have a neighbor B as an active next hop for destination D, while B has invalidated the route to D.
   */
  Time m_deletePeriod;
  Time m_nextHopWait; ///< Period of our waiting for the neighbour's RREP_ACK
  Time m_blackListTimeout; ///< Time for which the node is put into the blacklist
  uint32_t m_maxQueueLen; ///< The maximum number of packets that we allow a routing protocol to buffer.
  Time m_maxQueueTime; ///< The maximum period of time that a routing protocol is allowed to buffer a packet
      ↪ for.
  bool m_destinationOnly; ///< Indicates only the destination may respond to this RREQ.
  bool m_gratuitousReply; ///< Indicates whether a gratuitous RREP should be unicast to the node originated
      ↪ route discovery.
  bool m_enableHello; ///< Indicates whether a hello messages enable
  bool m_enableBroadcast; ///< Indicates whether a a broadcast data packets forwarding enable
  //\}

  /// IP protocol
  Ptr<Ipv4> m_ipv4;
  /// Raw unicast socket per each IP interface, map socket -> iface address (IP + mask)
  std::map< Ptr<Socket>, Ipv4InterfaceAddress > m_socketAddresses;
  /// Raw subnet directed broadcast socket per each IP interface, map socket -> iface address (IP + mask)
  std::map< Ptr<Socket>, Ipv4InterfaceAddress > m_socketSubnetBroadcastAddresses;
  /// Loopback device used to defer RREQ until packet will be fully formed
  Ptr<NetDevice> m_lo;

  /// Routing table
  RoutingTable m_routingTable;
  /// A "drop-front" queue used by the routing layer to buffer packets to which it does not have a route.
  RequestQueue m_queue;
  /// Broadcast ID
  uint32_t m_requestId;
  /// Request sequence number
  uint32_t m_seqNo;
```

```
253   /// Handle duplicated RREQ
254   IdCache m_rreqIdCache;
255   /// Handle duplicated broadcast/multicast packets
256   DuplicatePacketDetection m_dpd;
257   /// Handle neighbors
258   Neighbors m_nb;
259   /// Number of RREQs used for RREQ rate control
260   uint16_t m_rreqCount;
261   /// Number of RERRs used for RERR rate control
262   uint16_t m_rerrCount;
263   // EA-AODV
264   uint16_t m_rrepCount;
265 public:
266   // EA-AODV Instance Variables
267   double m_energyLevel;  // Remaining power level of instance node
268   double transDistance;  // Transmition distance of instance node
269   float nodeX;  // Instance node X coordinate
270   float nodeY; // Instance node Y coordinate
271   Time m_lastUpdateTime;
272   Ipv4Address ipv4Addr;
273   bool isReachable;
274 private:
275   /// Start protocol operation
276   void Start ();
277   /**
278    * Queue packet and send route request
279    *
280    * \param p the packet to route
281    * \param header the IP header
282    * \param ucb the UnicastForwardCallback function
283    * \param ecb the ErrorCallback function
284    */
285   void DeferredRouteOutput (Ptr<const Packet> p, const Ipv4Header & header, UnicastForwardCallback ucb,
       ↪ ErrorCallback ecb);
286   /**
287    * If route exists and is valid, forward packet.
288    *
289    * \param p the packet to route
290    * \param header the IP header
291    * \param ucb the UnicastForwardCallback function
292    * \param ecb the ErrorCallback function
293    * \returns true if forwarded
294    */
295   bool Forwarding (Ptr<const Packet> p, const Ipv4Header & header, UnicastForwardCallback ucb, ErrorCallback
       ↪ ecb);
296   /**
297    * Repeated attempts by a source node at route discovery for a single destination
298    * use the expanding ring search technique.
299    * \param dst the destination IP address
300    */
301   void ScheduleRreqRetry (Ipv4Address dst);
302   /**
303    * Set lifetime field in routing table entry to the maximum of existing lifetime and lt, if the entry exists
304    * \param addr - destination address
305    * \param lt - proposed time for lifetime field in routing table entry for destination with address addr.
306    * \return true if route to destination address addr exist
307    */
308   bool UpdateRouteLifeTime (Ipv4Address addr, Time lt);
309   /**
310    * Update neighbor record.
311    * \param receiver is supposed to be my interface
312    * \param sender is supposed to be IP address of my neighbor.
313    */
314   void UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver);
315   /**
316    * Test whether the provided address is assigned to an interface on this node
317    * \param src the source IP address
318    * \returns true if the IP address is the node's IP address
```

```
319      */
320     bool IsMyOwnAddress (Ipv4Address src);
321     /**
322      * Find unicast socket with local interface address iface
323      *
324      * \param iface the interface
325      * \returns the socket associated with the interface
326      */
327     Ptr<Socket> FindSocketWithInterfaceAddress (Ipv4InterfaceAddress iface) const;
328     /**
329      * Find subnet directed broadcast socket with local interface address iface
330      *
331      * \param iface the interface
332      * \returns the socket associated with the interface
333      */
334     Ptr<Socket> FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress iface) const;
335     /**
336      * Process hello message
337      *
338      * \param rrepHeader RREP message header
339      * \param receiverIfaceAddr receiver interface IP address
340      */
341     void ProcessHello (RrepHeader const & rrepHeader, Ipv4Address receiverIfaceAddr);
342     /**
343      * Create loopback route for given header
344      *
345      * \param header the IP header
346      * \param oif the output interface net device
347      * \returns the route
348      */
349     Ptr<Ipv4Route> LoopbackRoute (const Ipv4Header & header, Ptr<NetDevice> oif) const;
350
351     ///\name Receive control packets
352     //\{
353     /// Receive and process control packet
354     void RecvEaAodv (Ptr<Socket> socket);
355     /// Receive RREQ
356     void RecvRequest (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src);
357     /// Receive RREP
358     void RecvReply (Ptr<Packet> p, Ipv4Address my,Ipv4Address src);
359     /// Receive RREP_ACK
360     void RecvReplyAck (Ipv4Address neighbor);
361     /// Receive RERR from node with address src
362     void RecvError (Ptr<Packet> p, Ipv4Address src);
363     //\}
364
365     ///\name Send
366     //\{
367     /// Forward packet from route request queue
368     void SendPacketFromQueue (Ipv4Address dst, Ptr<Ipv4Route> route);
369     /// Send hello
370     void SendHello ();
371     /// Send RREQ
372     void SendRequest (Ipv4Address dst);
373     /// Send RREP
374     void SendReply (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin);
375     /** Send RREP by intermediate node
376      * \param toDst routing table entry to destination
377      * \param toOrigin routing table entry to originator
378      * \param gratRep indicates whether a gratuitous RREP should be unicast to destination
379      */
380     void SendReplyByIntermediateNode (RoutingTableEntry & toDst, RoutingTableEntry & toOrigin, bool gratRep);
381     /// Send RREP_ACK
382     void SendReplyAck (Ipv4Address neighbor);
383     /// Initiate RERR
384     void SendRerrWhenBreaksLinkToNextHop (Ipv4Address nextHop);
385     /// Forward RERR
386     void SendRerrMessage (Ptr<Packet> packet, std::vector<Ipv4Address> precursors);
```

```
387   /**
388    * Send RERR message when no route to forward input packet. Unicast if there is reverse route to
          ↪ originating node, broadcast otherwise.
389    * \param dst - destination node IP address
390    * \param dstSeqNo - destination node sequence number
391    * \param origin - originating node IP address
392    */
393   void SendRerrWhenNoRouteToForward (Ipv4Address dst, uint32_t dstSeqNo, Ipv4Address origin);
394   /// @}
395
396   /**
397    * Send packet to destination scoket
398    * \param socket - destination node socket
399    * \param packet - packet to send
400    * \param destination - destination node IP address
401    */
402   void SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address destination);
403
404   /// Hello timer
405   Timer m_htimer;
406   /// Schedule next send of hello message
407   void HelloTimerExpire ();
408   /// RREQ rate limit timer
409   Timer m_rreqRateLimitTimer;
410   /// Reset RREQ count and schedule RREQ rate limit timer with delay 1 sec.
411   void RreqRateLimitTimerExpire ();
412   /// RERR rate limit timer
413   Timer m_rerrRateLimitTimer;
414   /// Reset RERR count and schedule RERR rate limit timer with delay 1 sec.
415   void RerrRateLimitTimerExpire ();
416   /// Map IP address + RREQ timer.
417   std::map<Ipv4Address, Timer> m_addressReqTimer;
418
419   // EA-AODV begin
420   Timer m_powerDecayTimer;
421   void PowerDecayTimerExpire();
422   /**
423    * Handle route discovery process
424    * \param dst the destination IP address
425    */
426   void RouteRequestTimerExpire (Ipv4Address dst);
427   /**
428    * Mark link to neighbor node as unidirectional for blacklistTimeout
429    *
430    * \param neighbor the IP address of the neightbor node
431    * \param blacklistTimeout the black list timeout time
432    */
433   void AckTimerExpire (Ipv4Address neighbor, Time blacklistTimeout);
434
435   /// Provides uniform random variables.
436   Ptr<UniformRandomVariable> m_uniformRandomVariable;
437   /// Keep track of the last bcast time
438   Time m_lastBcastTime;
439   void getCoordinates(void);
440   float GetDistanceBetweenNodes(RoutingTableEntry route);
441 };
442
443 } //namespace eaAodv
444 } //namespace ns3
445
446 #endif /* EA_AODVROUTINGPROTOCOL_H */
```

ea-aodv-routing-protocol.cc

```
1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  /*
```

```
 * Copyright (c) 2009 IITP RAS
 *
 * 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
 *
 * Based on
 * NS-2 AODV model developed by the CMU/MONARCH group and optimized and
 * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
 *
 * AODV-UU implementation by Erik Nordstrm of Uppsala University
 * http://core.it.uu.se/core/index.php/AODV-UU
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */
#define NS_LOG_APPEND_CONTEXT \
  if (m_ipv4) { std::clog << "[node␣" << m_ipv4->GetObject<Node> ()->GetId () << "]␣"; }

#include "ea-aodv-routing-protocol.h"
#include "ns3/simulator.h"
#include "ns3/command-line.h"
#include "ns3/config.h"
#include "ns3/log.h"
#include "ns3/boolean.h"
#include "ns3/random-variable-stream.h"
#include "ns3/inet-socket-address.h"
#include "ns3/trace-source-accessor.h"
#include "ns3/udp-socket-factory.h"
#include "ns3/udp-l4-protocol.h"
#include "ns3/udp-header.h"
#include "ns3/wifi-net-device.h"
#include "ns3/adhoc-wifi-mac.h"
#include "ns3/mobility-model.h"
#include "ns3/string.h"
#include "ns3/pointer.h"
#include <algorithm>
#include <limits>
#include <cmath>

#define RREQ_WAIT_TIME 1

/* Debug and Simulation cout macros */
//#define DEBUG 1
#define SIMSTATS 1

#ifdef DEBUG
#define BUGOUT(x) do{std::cout<< x <<std::endl;}while(0)
#else
#define BUGOUT(x) do{}while(0)
#endif

#ifdef SIMSTATS
#define SIMOUT(x) do{std::cout<< x <<std::endl;}while(0)
#else
#define SIMOUT(x) do{}while(0)
#endif

// ENERGY MODEL CONSTANTS
```

```
#define ALPHA_TX  0.371354558f
#define MIN_RX_POW 1.58489319e-13f
#define LAMBDA2   0.015625f
#define PI2    9.869604401f
namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("EaAodvRoutingProtocol");

namespace eaAodv {
NS_OBJECT_ENSURE_REGISTERED (RoutingProtocol);

/// UDP Port for EA_AODV control traffic
const uint32_t RoutingProtocol::EA_AODV_PORT = 654;

// Energy Model
const float RoutingProtocol::power_receive = 0.075 * ESP32_VOLTAGE;
const float RoutingProtocol::power_active = 0.0225 * ESP32_VOLTAGE;
// based on single-core @ 80 MHz with modem-sleep
const float RoutingProtocol::m_aliveThreshold = 0.001;
const double RoutingProtocol::TxConst=(double)(PACKET_SIZE*8)/1e6 * ESP32_VOLTAGE *
  MIN_RX_POW * 16 * PI2 / (ALPHA_TX * LAMBDA2);
/** determined by:
 * ALPHA_TX is proportional constant relating output antenna power to current
 * consumption: P_tx(W) = I_tx * ALPHA_TX
 * E_tx = P_tx * time
 * E_tx = #bits/(802.11b bitrate) * ESP32_VOLTAGE * I_tx
 * E_tx = #bits/(802.11b bitrate) * ESP32_VOLTAGE * MIN_RX_POW * 16*pi^2 * distance^2
 *  / (ALPHA_TX * LAMBDA^2)
 **/
/**
* \ingroup eaAodv
* \brief Tag used by AODV implementation
*/
class DeferredRouteOutputTag : public Tag
{

public:
  /**
   * \brief Constructor
   * \param o the output interface
   */
  DeferredRouteOutputTag (int32_t o = -1) : Tag (),
                                            m_oif (o)
  {
  }

  /**
   * \brief Get the type ID.
   * \return the object TypeId
   */
  static TypeId GetTypeId ()
  {
    static TypeId tid = TypeId ("ns3::eaAodv::DeferredRouteOutputTag")
      .SetParent<Tag> ()
      .SetGroupName ("eaAodv")
      .AddConstructor<DeferredRouteOutputTag> ()
    ;
    return tid;
  }

  TypeId GetInstanceTypeId () const
  {
    return GetTypeId ();
  }

  /**
   * \brief Get the output interface
   * \return the output interface
```

```
139     */
140   int32_t GetInterface () const
141   {
142     return m_oif;
143   }
144 
145   /**
146    * \brief Set the output interface
147    * \param oif the output interface
148    */
149   void SetInterface (int32_t oif)
150   {
151     m_oif = oif;
152   }
153 
154   uint32_t GetSerializedSize () const
155   {
156     return sizeof(int32_t);
157   }
158 
159   void Serialize (TagBuffer i) const
160   {
161     i.WriteU32 (m_oif);
162   }
163 
164   void Deserialize (TagBuffer i)
165   {
166     m_oif = i.ReadU32 ();
167   }
168 
169   void Print (std::ostream &os) const
170   {
171     os << "DeferredRouteOutputTag:␣output␣interface␣=␣" << m_oif;
172   }
173 
174 private:
175   /// Positive if output device is fixed in RouteOutput
176   int32_t m_oif;
177 };
178 
179 NS_OBJECT_ENSURE_REGISTERED (DeferredRouteOutputTag);
180 
181 
182 //-----------------------------------------------------------------------------
183 RoutingProtocol::RoutingProtocol ()
184   : m_rreqRetries (2),
185     m_ttlStart (1),
186     m_ttlIncrement (2),
187     m_ttlThreshold (7),
188     m_timeoutBuffer (2),
189     m_rreqRateLimit (10),
190     m_rerrRateLimit (10),
191     m_activeRouteTimeout (Seconds (3)),
192     m_netDiameter (35),
193     m_nodeTraversalTime (MilliSeconds (40)),
194     m_netTraversalTime (Time ((2 * m_netDiameter) * m_nodeTraversalTime)),
195     m_pathDiscoveryTime ( Time (2 * m_netTraversalTime)),
196     m_myRouteTimeout (Time (2 * std::max (m_pathDiscoveryTime, m_activeRouteTimeout))),
197     m_helloInterval (Seconds (1)),
198     m_allowedHelloLoss (2),
199     m_deletePeriod (Time (5 * std::max (m_activeRouteTimeout, m_helloInterval))),
200     m_nextHopWait (m_nodeTraversalTime + MilliSeconds (10)),
201     m_blackListTimeout (Time (m_rreqRetries * m_netTraversalTime)),
202     m_maxQueueLen (64),
203     m_maxQueueTime (Seconds (30)),
204     m_destinationOnly (false),
205     m_gratuitousReply (true),
206     m_enableHello (false),
```

```
207     m_routingTable (m_deletePeriod),
208     m_queue (m_maxQueueLen, m_maxQueueTime),
209     m_requestId (0),
210     m_seqNo (0),
211     m_rreqIdCache (m_pathDiscoveryTime),
212     m_dpd (m_pathDiscoveryTime),
213     m_nb (m_helloInterval),
214     m_rreqCount (0),
215     m_rerrCount (0),
216     m_rrepCount (0),
217     m_energyLevel(1000.0),
218     transDistance(250.0),
219     nodeX(-1),
220     nodeY(-1),
221     m_lastUpdateTime(Seconds(0.0)),
222     ipv4Addr(Ipv4Address::GetAny()),
223     isReachable(true),
224     m_htimer (Timer::CANCEL_ON_DESTROY),
225     m_rreqRateLimitTimer (Timer::CANCEL_ON_DESTROY),
226     m_rerrRateLimitTimer (Timer::CANCEL_ON_DESTROY),
227     m_powerDecayTimer(Timer::CANCEL_ON_DESTROY),
228     m_lastBcastTime (Seconds (0))
229 {
230   m_nb.SetCallback (MakeCallback (&RoutingProtocol::SendRerrWhenBreaksLinkToNextHop, this));
231 }
232
233 TypeId
234 RoutingProtocol::GetTypeId (void)
235 {
236   static TypeId tid = TypeId ("ns3::eaAodv::RoutingProtocol")
237     .SetParent<Ipv4RoutingProtocol> ()
238     .SetGroupName ("eaAodv")
239     .AddConstructor<RoutingProtocol> ()
240     .AddAttribute ("HelloInterval", "HELLO␣messages␣emission␣interval.",
241                    TimeValue (Seconds (1)),
242                    MakeTimeAccessor (&RoutingProtocol::m_helloInterval),
243                    MakeTimeChecker ())
244     .AddAttribute ("TtlStart", "Initial␣TTL␣value␣for␣RREQ.",
245                    UintegerValue (1),
246                    MakeUintegerAccessor (&RoutingProtocol::m_ttlStart),
247                    MakeUintegerChecker<uint16_t> ())
248     .AddAttribute ("TtlIncrement", "TTL␣increment␣for␣each␣attempt␣using␣the␣expanding␣ring␣search␣for␣RREQ␣
         ↪ dissemination.",
249                    UintegerValue (2),
250                    MakeUintegerAccessor (&RoutingProtocol::m_ttlIncrement),
251                    MakeUintegerChecker<uint16_t> ())
252     .AddAttribute ("TtlThreshold", "Maximum␣TTL␣value␣for␣expanding␣ring␣search,␣TTL␣=␣NetDiameter␣is␣used␣
         ↪ beyond␣this␣value.",
253                    UintegerValue (7),
254                    MakeUintegerAccessor (&RoutingProtocol::m_ttlThreshold),
255                    MakeUintegerChecker<uint16_t> ())
256     .AddAttribute ("TimeoutBuffer", "Provide␣a␣buffer␣for␣the␣timeout.",
257                    UintegerValue (2),
258                    MakeUintegerAccessor (&RoutingProtocol::m_timeoutBuffer),
259                    MakeUintegerChecker<uint16_t> ())
260     .AddAttribute ("RreqRetries", "Maximum␣number␣of␣retransmissions␣of␣RREQ␣to␣discover␣a␣route",
261                    UintegerValue (2),
262                    MakeUintegerAccessor (&RoutingProtocol::m_rreqRetries),
263                    MakeUintegerChecker<uint32_t> ())
264     .AddAttribute ("RreqRateLimit", "Maximum␣number␣of␣RREQ␣per␣second.",
265                    UintegerValue (10),
266                    MakeUintegerAccessor (&RoutingProtocol::m_rreqRateLimit),
267                    MakeUintegerChecker<uint32_t> ())
268     .AddAttribute ("RerrRateLimit", "Maximum␣number␣of␣RERR␣per␣second.",
269                    UintegerValue (10),
270                    MakeUintegerAccessor (&RoutingProtocol::m_rerrRateLimit),
271                    MakeUintegerChecker<uint32_t> ())
272     .AddAttribute ("NodeTraversalTime", "Conservative␣estimate␣of␣the␣average␣one␣hop␣traversal␣time␣for␣
```

```
        ↪ packets␣and␣should␣include␣"
273                 "queuing␣delays,␣interrupt␣processing␣times␣and␣transfer␣times.",
274                 TimeValue (MilliSeconds (40)),
275                 MakeTimeAccessor (&RoutingProtocol::m_nodeTraversalTime),
276                 MakeTimeChecker ())
277     .AddAttribute ("NextHopWait", "Period␣of␣our␣waiting␣for␣the␣neighbour's␣RREP_ACK␣=␣10␣ms␣+␣
        ↪ NodeTraversalTime",
278                 TimeValue (MilliSeconds (50)),
279                 MakeTimeAccessor (&RoutingProtocol::m_nextHopWait),
280                 MakeTimeChecker ())
281     .AddAttribute ("ActiveRouteTimeout", "Period␣of␣time␣during␣which␣the␣route␣is␣considered␣to␣be␣valid",
282                 TimeValue (Seconds (3)),
283                 MakeTimeAccessor (&RoutingProtocol::m_activeRouteTimeout),
284                 MakeTimeChecker ())
285     .AddAttribute ("MyRouteTimeout", "Value␣of␣lifetime␣field␣in␣RREP␣generating␣by␣this␣node␣=␣2␣*␣
        ↪ max(ActiveRouteTimeout,␣PathDiscoveryTime)",
286                 TimeValue (Seconds (11.2)),
287                 MakeTimeAccessor (&RoutingProtocol::m_myRouteTimeout),
288                 MakeTimeChecker ())
289     .AddAttribute ("BlackListTimeout", "Time␣for␣which␣the␣node␣is␣put␣into␣the␣blacklist␣=␣RreqRetries␣*␣
        ↪ NetTraversalTime",
290                 TimeValue (Seconds (5.6)),
291                 MakeTimeAccessor (&RoutingProtocol::m_blackListTimeout),
292                 MakeTimeChecker ())
293     .AddAttribute ("DeletePeriod", "DeletePeriod␣is␣intended␣to␣provide␣an␣upper␣bound␣on␣the␣time␣for␣which␣
        ↪ an␣upstream␣node␣A␣"
294                 "can␣have␣a␣neighbor␣B␣as␣an␣active␣next␣hop␣for␣destination␣D,␣while␣B␣has␣invalidated␣the␣
                    ↪ route␣to␣D."
295                 "␣=␣5␣*␣max␣(HelloInterval,␣ActiveRouteTimeout)",
296                 TimeValue (Seconds (15)),
297                 MakeTimeAccessor (&RoutingProtocol::m_deletePeriod),
298                 MakeTimeChecker ())
299     .AddAttribute ("NetDiameter", "Net␣diameter␣measures␣the␣maximum␣possible␣number␣of␣hops␣between␣two␣
        ↪ nodes␣in␣the␣network",
300                 UintegerValue (35),
301                 MakeUintegerAccessor (&RoutingProtocol::m_netDiameter),
302                 MakeUintegerChecker<uint32_t> ())
303     .AddAttribute ("NetTraversalTime", "Estimate␣of␣the␣average␣net␣traversal␣time␣=␣2␣*␣NodeTraversalTime␣*␣
        ↪ NetDiameter",
304                 TimeValue (Seconds (2.8)),
305                 MakeTimeAccessor (&RoutingProtocol::m_netTraversalTime),
306                 MakeTimeChecker ())
307     .AddAttribute ("PathDiscoveryTime", "Estimate␣of␣maximum␣time␣needed␣to␣find␣route␣in␣network␣=␣2␣*␣
        ↪ NetTraversalTime",
308                 TimeValue (Seconds (5.6)),
309                 MakeTimeAccessor (&RoutingProtocol::m_pathDiscoveryTime),
310                 MakeTimeChecker ())
311     .AddAttribute ("MaxQueueLen", "Maximum␣number␣of␣packets␣that␣we␣allow␣a␣routing␣protocol␣to␣buffer.",
312                 UintegerValue (64),
313                 MakeUintegerAccessor (&RoutingProtocol::SetMaxQueueLen,
314                                       &RoutingProtocol::GetMaxQueueLen),
315                 MakeUintegerChecker<uint32_t> ())
316     .AddAttribute ("MaxQueueTime", "Maximum␣time␣packets␣can␣be␣queued␣(in␣seconds)",
317                 TimeValue (Seconds (30)),
318                 MakeTimeAccessor (&RoutingProtocol::SetMaxQueueTime,
319                                   &RoutingProtocol::GetMaxQueueTime),
320                 MakeTimeChecker ())
321     .AddAttribute ("AllowedHelloLoss", "Number␣of␣hello␣messages␣which␣may␣be␣loss␣for␣valid␣link.",
322                 UintegerValue (2),
323                 MakeUintegerAccessor (&RoutingProtocol::m_allowedHelloLoss),
324                 MakeUintegerChecker<uint16_t> ())
325     .AddAttribute ("GratuitousReply", "Indicates␣whether␣a␣gratuitous␣RREP␣should␣be␣unicast␣to␣the␣node␣
        ↪ originated␣route␣discovery.",
326                 BooleanValue (true),
327                 MakeBooleanAccessor (&RoutingProtocol::SetGratuitousReplyFlag,
328                                      &RoutingProtocol::GetGratuitousReplyFlag),
329                 MakeBooleanChecker ())
330     .AddAttribute ("DestinationOnly", "Indicates␣only␣the␣destination␣may␣respond␣to␣this␣RREQ.",
```

```
                   BooleanValue (false),
                   MakeBooleanAccessor (&RoutingProtocol::SetDestinationOnlyFlag,
                                        &RoutingProtocol::GetDestinationOnlyFlag),
                   MakeBooleanChecker ())
    .AddAttribute ("EnableHello", "Indicates whether a hello messages enable.",
                   BooleanValue (true),
                   MakeBooleanAccessor (&RoutingProtocol::SetHelloEnable,
                                        &RoutingProtocol::GetHelloEnable),
                   MakeBooleanChecker ())
    .AddAttribute ("EnableBroadcast", "Indicates whether a broadcast data packets forwarding enable.",
                   BooleanValue (true),
                   MakeBooleanAccessor (&RoutingProtocol::SetBroadcastEnable,
                                        &RoutingProtocol::GetBroadcastEnable),
                   MakeBooleanChecker ())
    .AddAttribute ("UniformRv",
                   "Access to the underlying UniformRandomVariable",
                   StringValue ("ns3::UniformRandomVariable"),
                   MakePointerAccessor (&RoutingProtocol::m_uniformRandomVariable),
                   MakePointerChecker<UniformRandomVariable> ())
    .AddAttribute ("MaxPower",
                   "Maximum battery power of a node",
                   DoubleValue(100),
                   MakeDoubleAccessor (&RoutingProtocol::m_energyLevel),
                   MakeDoubleChecker<double> ())
    .AddAttribute ("TransDistance",
                   "Transmission Distance for AODV algroithm",
                   DoubleValue(250),
                   MakeDoubleAccessor (&RoutingProtocol::transDistance),
                   MakeDoubleChecker<double> ())
  ;
  return tid;
}

void
RoutingProtocol::SetMaxQueueLen (uint32_t len)
{
  m_maxQueueLen = len;
  m_queue.SetMaxQueueLen (len);
}
void
RoutingProtocol::SetMaxQueueTime (Time t)
{
  m_maxQueueTime = t;
  m_queue.SetQueueTimeout (t);
}

RoutingProtocol::~RoutingProtocol ()
{
}

void
RoutingProtocol::DoDispose ()
{
  if (m_energyLevel > m_aliveThreshold) {
    SIMOUT("Node "<<ipv4Addr<<" has "<<m_energyLevel<<" remaining power.");
  }
  m_ipv4 = 0;
  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::iterator iter =
         m_socketAddresses.begin (); iter != m_socketAddresses.end (); iter++)
    {
      iter->first->Close ();
    }
  m_socketAddresses.clear ();
  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::iterator iter =
         m_socketSubnetBroadcastAddresses.begin (); iter != m_socketSubnetBroadcastAddresses.end (); iter++)
    {
      iter->first->Close ();
    }
```

```
399   m_socketSubnetBroadcastAddresses.clear ();
400   Ipv4RoutingProtocol::DoDispose ();
401 }
402
403 void
404 RoutingProtocol::PrintRoutingTable (Ptr<OutputStreamWrapper> stream, Time::Unit unit) const
405 {
406   *stream->GetStream () << "Node: " << m_ipv4->GetObject<Node> ()->GetId ()
407                         << "; Time: " << Now ().As (unit)
408                         << ", Local time: " << GetObject<Node> ()->GetLocalTime ().As (unit)
409                         << ", AODV Routing table" << std::endl;
410
411   m_routingTable.Print (stream);
412   *stream->GetStream () << std::endl;
413 }
414
415 int64_t
416 RoutingProtocol::AssignStreams (int64_t stream)
417 {
418   NS_LOG_FUNCTION (this << stream);
419   m_uniformRandomVariable->SetStream (stream);
420   return 1;
421 }
422
423 void
424 RoutingProtocol::Start ()
425 {
426   NS_LOG_FUNCTION (this);
427   if (m_enableHello)
428     {
429       m_nb.ScheduleTimer ();
430     }
431   m_rreqRateLimitTimer.SetFunction (&RoutingProtocol::RreqRateLimitTimerExpire,
432                                     this);
433   m_rreqRateLimitTimer.Schedule (Seconds (1));
434
435   m_rerrRateLimitTimer.SetFunction (&RoutingProtocol::RerrRateLimitTimerExpire,
436                                     this);
437   m_rerrRateLimitTimer.Schedule (Seconds (1));
438
439   // EA-AODV set timer for energy decay
440   m_powerDecayTimer.SetFunction (&RoutingProtocol::PowerDecayTimerExpire, this);
441   m_powerDecayTimer.Schedule(MilliSeconds(2));
442
443   ipv4Addr = m_ipv4->GetAddress (1, 0).GetLocal ();
444   getCoordinates();
445   SIMOUT("Node created "<<ipv4Addr<<" at pos: ("<<nodeX<<","<<nodeY<<") with initial power "<<m_energyLevel);
446 }
447
448 Ptr<Ipv4Route>
449 RoutingProtocol::RouteOutput (Ptr<Packet> p, const Ipv4Header &header,
450                               Ptr<NetDevice> oif, Socket::SocketErrno &sockerr)
451 {
452   NS_LOG_FUNCTION (this << header << (oif ? oif->GetIfIndex () : 0));
453   if (!p)
454     {
455       NS_LOG_DEBUG ("Packet is == 0");
456       return LoopbackRoute (header, oif); // later
457     }
458   if (m_socketAddresses.empty ())
459     {
460       sockerr = Socket::ERROR_NOROUTETOHOST;
461       NS_LOG_LOGIC ("No aodv interfaces");
462       Ptr<Ipv4Route> route;
463       return route;
464     }
465   sockerr = Socket::ERROR_NOTERROR;
466   Ptr<Ipv4Route> route;
```

```
  Ipv4Address dst = header.GetDestination ();
  RoutingTableEntry rt;
  if (m_routingTable.LookupValidRoute (dst, rt))
    {
      route = rt.GetRoute ();
      NS_ASSERT (route != 0);
      NS_LOG_DEBUG ("Exist route to " << route->GetDestination () << " from interface " << route->GetSource
          ↪ ());
      if (oif != 0 && route->GetOutputDevice () != oif)
        {
          NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
          sockerr = Socket::ERROR_NOROUTETOHOST;
          return Ptr<Ipv4Route> ();
        }
      UpdateRouteLifeTime (dst, m_activeRouteTimeout);
      UpdateRouteLifeTime (route->GetGateway (), m_activeRouteTimeout);
      // EA-AODV POWER DECAY
      BUGOUT(ipv4Addr<<": power decayed for transmission.");
      double nextHopDistance = powf(GetDistanceBetweenNodes(rt),2);
      m_energyLevel -= TxConst * powf(nextHopDistance,2);
      return route;
    }

  // Valid route not found, in this case we return loopback.
  // Actual route request will be deferred until packet will be fully formed,
  // routed to loopback, received from loopback and passed to RouteInput (see below)
  uint32_t iif = (oif ? m_ipv4->GetInterfaceForDevice (oif) : -1);
  DeferredRouteOutputTag tag (iif);
  NS_LOG_DEBUG ("Valid Route not found");
  if (!p->PeekPacketTag (tag))
    {
      p->AddPacketTag (tag);
    }
  return LoopbackRoute (header, oif);
}

void
RoutingProtocol::DeferredRouteOutput (Ptr<const Packet> p, const Ipv4Header & header,
                                      UnicastForwardCallback ucb, ErrorCallback ecb)
{
  NS_LOG_FUNCTION (this << p << header);
  NS_ASSERT (p != 0 && p != Ptr<Packet> ());
  if (m_energyLevel > m_aliveThreshold) {
    QueueEntry newEntry (p, header, ucb, ecb);
    bool result = m_queue.Enqueue (newEntry);
    if (result)
      {
        NS_LOG_LOGIC ("Add packet " << p->GetUid () << " to queue. Protocol " << (uint16_t) header.GetProtocol
            ↪ ());
        RoutingTableEntry rt;
        bool result = m_routingTable.LookupRoute (header.GetDestination (), rt);
        if (!result || ((rt.GetFlag () != IN_SEARCH) && result))
          {
            NS_LOG_LOGIC ("Send new RREQ for outbound packet to " << header.GetDestination ());
            SendRequest (header.GetDestination ());
          }
      }
  }
}

bool
RoutingProtocol::RouteInput (Ptr<const Packet> p, const Ipv4Header &header,
                             Ptr<const NetDevice> idev, UnicastForwardCallback ucb,
                             MulticastForwardCallback mcb, LocalDeliverCallback lcb, ErrorCallback ecb)
{
  NS_LOG_FUNCTION (this << p->GetUid () << header.GetDestination () << idev->GetAddress ());
  if (m_energyLevel > m_aliveThreshold) {
    if (m_socketAddresses.empty ())
```

```
533       {
534         NS_LOG_LOGIC ("No␣aodv␣interfaces");
535         return false;
536       }
537     NS_ASSERT (m_ipv4 != 0);
538     NS_ASSERT (p != 0);
539     // Check if input device supports IP
540     NS_ASSERT (m_ipv4->GetInterfaceForDevice (idev) >= 0);
541     int32_t iif = m_ipv4->GetInterfaceForDevice (idev);
542
543     Ipv4Address dst = header.GetDestination ();
544     Ipv4Address origin = header.GetSource ();
545
546     // Deferred route request
547     if (idev == m_lo)
548       {
549         DeferredRouteOutputTag tag;
550         if (p->PeekPacketTag (tag))
551           {
552             DeferredRouteOutput (p, header, ucb, ecb);
553             return true;
554           }
555       }
556
557     // Duplicate of own packet
558     if (IsMyOwnAddress (origin))
559       {
560         return true;
561       }
562
563     // AODV is not a multicast routing protocol
564     if (dst.IsMulticast ())
565       {
566         return false;
567       }
568
569     // Broadcast local delivery/forwarding
570     for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
571           m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
572       {
573         Ipv4InterfaceAddress iface = j->second;
574         if (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()) == iif)
575           {
576             if (dst == iface.GetBroadcast () || dst.IsBroadcast ())
577               {
578                 if (m_dpd.IsDuplicate (p, header))
579                   {
580                     NS_LOG_DEBUG ("Duplicated␣packet␣" << p->GetUid () << "␣from␣" << origin << ".␣Drop.");
581                     return true;
582                   }
583                 UpdateRouteLifeTime (origin, m_activeRouteTimeout);
584                 Ptr<Packet> packet = p->Copy ();
585                 if (lcb.IsNull () == false)
586                   {
587                     NS_LOG_LOGIC ("Broadcast␣local␣delivery␣to␣" << iface.GetLocal ());
588                     lcb (p, header, iif);
589                     // Fall through to additional processing
590                   }
591                 else
592                   {
593                     NS_LOG_ERROR ("Unable␣to␣deliver␣packet␣locally␣due␣to␣null␣callback␣" << p->GetUid () << "␣
                          ↪ from␣" << origin);
594                     ecb (p, header, Socket::ERROR_NOROUTETOHOST);
595                   }
596                 if (!m_enableBroadcast)
597                   {
598                     return true;
599                   }
```

```
600             if (header.GetProtocol () == UdpL4Protocol::PROT_NUMBER)
601               {
602                 UdpHeader udpHeader;
603                 p->PeekHeader (udpHeader);
604                 if (udpHeader.GetDestinationPort () == EA_AODV_PORT)
605                   {
606                     // AODV packets sent in broadcast are already managed
607                     return true;
608                   }
609               }
610             if (header.GetTtl () > 1)
611               {
612                 NS_LOG_LOGIC ("Forward␣broadcast.␣TTL␣" << (uint16_t) header.GetTtl ());
613                 RoutingTableEntry toBroadcast;
614                 if (m_routingTable.LookupRoute (dst, toBroadcast))
615                   {
616                     Ptr<Ipv4Route> route = toBroadcast.GetRoute ();
617                     ucb (route, packet, header);
618                   }
619                 else
620                   {
621                     NS_LOG_DEBUG ("No␣route␣to␣forward␣broadcast.␣Drop␣packet␣" << p->GetUid ());
622                   }
623               }
624             else
625               {
626                 NS_LOG_DEBUG ("TTL␣exceeded.␣Drop␣packet␣" << p->GetUid ());
627               }
628             return true;
629           }
630       }
631     }
632
633   // Unicast local delivery
634   if (m_ipv4->IsDestinationAddress (dst, iif))
635     {
636       UpdateRouteLifeTime (origin, m_activeRouteTimeout);
637       RoutingTableEntry toOrigin;
638       if (m_routingTable.LookupValidRoute (origin, toOrigin))
639         {
640           UpdateRouteLifeTime (toOrigin.GetNextHop (), m_activeRouteTimeout);
641           m_nb.Update (toOrigin.GetNextHop (), m_activeRouteTimeout);
642         }
643       if (lcb.IsNull () == false)
644         {
645           NS_LOG_LOGIC ("Unicast␣local␣delivery␣to␣" << dst);
646           lcb (p, header, iif);
647         }
648       else
649         {
650           NS_LOG_ERROR ("Unable␣to␣deliver␣packet␣locally␣due␣to␣null␣callback␣" << p->GetUid () << "␣from␣"
                ↪ << origin);
651           ecb (p, header, Socket::ERROR_NOROUTETOHOST);
652         }
653       return true;
654     }
655
656   // Check if input device supports IP forwarding
657   if (m_ipv4->IsForwarding (iif) == false)
658     {
659       NS_LOG_LOGIC ("Forwarding␣disabled␣for␣this␣interface");
660       ecb (p, header, Socket::ERROR_NOROUTETOHOST);
661       return true;
662     }
663
664   // Forwarding
665   return Forwarding (p, header, ucb, ecb);
666 } else return false;
```

```
667 }
668
669 bool
670 RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
671                              UnicastForwardCallback ucb, ErrorCallback ecb)
672 {
673   NS_LOG_FUNCTION (this);
674   if (m_energyLevel > m_aliveThreshold) {
675     Ipv4Address dst = header.GetDestination ();
676     Ipv4Address origin = header.GetSource ();
677     m_routingTable.Purge ();
678     RoutingTableEntry toDst;
679     if (m_routingTable.LookupRoute (dst, toDst))
680       {
681         if (toDst.GetFlag () == VALID)
682           {
683             Ptr<Ipv4Route> route = toDst.GetRoute ();
684             NS_LOG_LOGIC (route->GetSource () << "␣forwarding␣to␣" << dst << "␣from␣" << origin << "␣packet␣"
                  ↪ << p->GetUid ());
685
686             /*
687              * Each time a route is used to forward a data packet, its Active Route
688              * Lifetime field of the source, destination and the next hop on the
689              * path to the destination is updated to be no less than the current
690              * time plus ActiveRouteTimeout.
691              */
692             UpdateRouteLifeTime (origin, m_activeRouteTimeout);
693             UpdateRouteLifeTime (dst, m_activeRouteTimeout);
694             UpdateRouteLifeTime (route->GetGateway (), m_activeRouteTimeout);
695             /*
696              * Since the route between each originator and destination pair is expected to be symmetric, the
697              * Active Route Lifetime for the previous hop, along the reverse path back to the IP source, is
                    ↪ also updated
698              * to be no less than the current time plus ActiveRouteTimeout
699              */
700             RoutingTableEntry toOrigin;
701             m_routingTable.LookupRoute (origin, toOrigin);
702             UpdateRouteLifeTime (toOrigin.GetNextHop (), m_activeRouteTimeout);
703
704             m_nb.Update (route->GetGateway (), m_activeRouteTimeout);
705             m_nb.Update (toOrigin.GetNextHop (), m_activeRouteTimeout);
706
707             ucb (route, p, header);
708             return true;
709           }
710         else
711           {
712             if (toDst.GetValidSeqNo ())
713               {
714                 SendRerrWhenNoRouteToForward (dst, toDst.GetSeqNo (), origin);
715                 NS_LOG_DEBUG ("Drop␣packet␣" << p->GetUid () << "␣because␣no␣route␣to␣forward␣it.");
716                 return false;
717               }
718           }
719       }
720     NS_LOG_LOGIC ("route␣not␣found␣to␣" << dst << ".␣Send␣RERR␣message.");
721     NS_LOG_DEBUG ("Drop␣packet␣" << p->GetUid () << "␣because␣no␣route␣to␣forward␣it.");
722     SendRerrWhenNoRouteToForward (dst, 0, origin);
723     return false;
724   } else return false;
725 }
726
727 void
728 RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
729 {
730   NS_ASSERT (ipv4 != 0);
731   NS_ASSERT (m_ipv4 == 0);
732
```

```
733   m_ipv4 = ipv4;
734
735   // Create lo route. It is asserted that the only one interface up for now is loopback
736   NS_ASSERT (m_ipv4->GetNInterfaces () == 1 && m_ipv4->GetAddress (0, 0).GetLocal () == Ipv4Address
          ↪ ("127.0.0.1"));
737   m_lo = m_ipv4->GetNetDevice (0);
738   NS_ASSERT (m_lo != 0);
739   // Remember lo route
740   RoutingTableEntry rt (/*device=*/ m_lo, /*dst=*/ Ipv4Address::GetLoopback (), /*know seqno=*/ true,
          ↪ /*seqno=*/ 0,
741                                 /*iface=*/ Ipv4InterfaceAddress (Ipv4Address::GetLoopback (), Ipv4Mask
                                      ↪ ("255.0.0.0")),
742                                 /*hops=*/ 1, /*next hop=*/ Ipv4Address::GetLoopback (),
743                                 /*lifetime=*/ Simulator::GetMaximumSimulationTime ());
744   m_routingTable.AddRoute (rt);
745
746   Simulator::ScheduleNow (&RoutingProtocol::Start, this);
747 }
748
749 void
750 RoutingProtocol::NotifyInterfaceUp (uint32_t i)
751 {
752   NS_LOG_FUNCTION (this << m_ipv4->GetAddress (i, 0).GetLocal ());
753   Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
754   if (l3->GetNAddresses (i) > 1)
755     {
756       NS_LOG_WARN ("EA_AODV␣does␣not␣work␣with␣more␣then␣one␣address␣per␣each␣interface.");
757     }
758   Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
759   if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
760     {
761       return;
762     }
763
764   // Create a socket to listen only on this interface
765   Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
766                                              UdpSocketFactory::GetTypeId ());
767   NS_ASSERT (socket != 0);
768   socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvEaAodv, this));
769   socket->BindToNetDevice (l3->GetNetDevice (i));
770   socket->Bind (InetSocketAddress (iface.GetLocal (), EA_AODV_PORT));
771   socket->SetAllowBroadcast (true);
772   socket->SetIpRecvTtl (true);
773   m_socketAddresses.insert (std::make_pair (socket, iface));
774
775   // create also a subnet broadcast socket
776   socket = Socket::CreateSocket (GetObject<Node> (),
777                                  UdpSocketFactory::GetTypeId ());
778   NS_ASSERT (socket != 0);
779   socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvEaAodv, this));
780   socket->BindToNetDevice (l3->GetNetDevice (i));
781   socket->Bind (InetSocketAddress (iface.GetBroadcast (), EA_AODV_PORT));
782   socket->SetAllowBroadcast (true);
783   socket->SetIpRecvTtl (true);
784   m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));
785
786   // Add local broadcast record to the routing table
787   Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
788   RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true, /*seqno=*/ 0,
          ↪ /*iface=*/ iface,
789                                 /*hops=*/ 1, /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
                                      ↪ Simulator::GetMaximumSimulationTime ());
790   m_routingTable.AddRoute (rt);
791
792   if (l3->GetInterface (i)->GetArpCache ())
793     {
794       m_nb.AddArpCache (l3->GetInterface (i)->GetArpCache ());
795     }
```

```
796
797   // Allow neighbor manager use this interface for layer 2 feedback if possible
798   Ptr<WifiNetDevice> wifi = dev->GetObject<WifiNetDevice> ();
799   if (wifi == 0)
800     {
801       return;
802     }
803   Ptr<WifiMac> mac = wifi->GetMac ();
804   if (mac == 0)
805     {
806       return;
807     }
808
809   mac->TraceConnectWithoutContext ("TxErrHeader", m_nb.GetTxErrorCallback ());
810 }
811
812 void
813 RoutingProtocol::NotifyInterfaceDown (uint32_t i)
814 {
815   NS_LOG_FUNCTION (this << m_ipv4->GetAddress (i, 0).GetLocal ());
816
817   // Disable layer 2 link state monitoring (if possible)
818   Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
819   Ptr<NetDevice> dev = l3->GetNetDevice (i);
820   Ptr<WifiNetDevice> wifi = dev->GetObject<WifiNetDevice> ();
821   if (wifi != 0)
822     {
823       Ptr<WifiMac> mac = wifi->GetMac ()->GetObject<AdhocWifiMac> ();
824       if (mac != 0)
825         {
826           mac->TraceDisconnectWithoutContext ("TxErrHeader",
827                                               m_nb.GetTxErrorCallback ());
828           m_nb.DelArpCache (l3->GetInterface (i)->GetArpCache ());
829         }
830     }
831
832   // Close socket
833   Ptr<Socket> socket = FindSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
834   NS_ASSERT (socket);
835   socket->Close ();
836   m_socketAddresses.erase (socket);
837
838   // Close socket
839   socket = FindSubnetBroadcastSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
840   NS_ASSERT (socket);
841   socket->Close ();
842   m_socketSubnetBroadcastAddresses.erase (socket);
843
844   if (m_socketAddresses.empty ())
845     {
846       NS_LOG_LOGIC ("No␣aodv␣interfaces");
847       m_htimer.Cancel ();
848       m_nb.Clear ();
849       m_routingTable.Clear ();
850       return;
851     }
852   m_routingTable.DeleteAllRoutesFromInterface (m_ipv4->GetAddress (i, 0));
853 }
854
855 void
856 RoutingProtocol::NotifyAddAddress (uint32_t i, Ipv4InterfaceAddress address)
857 {
858   NS_LOG_FUNCTION (this << "␣interface␣" << i << "␣address␣" << address);
859   Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
860   if (!l3->IsUp (i))
861     {
862       return;
863     }
```

```
  if (l3->GetNAddresses (i) == 1)
    {
      Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
      Ptr<Socket> socket = FindSocketWithInterfaceAddress (iface);
      if (!socket)
        {
          if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
            {
              return;
            }
          // Create a socket to listen only on this interface
          Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
                                                     UdpSocketFactory::GetTypeId ());
          NS_ASSERT (socket != 0);
          socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvEaAodv,this));
          socket->BindToNetDevice (l3->GetNetDevice (i));
          socket->Bind (InetSocketAddress (iface.GetLocal (), EA_AODV_PORT));
          socket->SetAllowBroadcast (true);
          m_socketAddresses.insert (std::make_pair (socket, iface));

          // create also a subnet directed broadcast socket
          socket = Socket::CreateSocket (GetObject<Node> (),
                                         UdpSocketFactory::GetTypeId ());
          NS_ASSERT (socket != 0);
          socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvEaAodv, this));
          socket->BindToNetDevice (l3->GetNetDevice (i));
          socket->Bind (InetSocketAddress (iface.GetBroadcast (), EA_AODV_PORT));
          socket->SetAllowBroadcast (true);
          socket->SetIpRecvTtl (true);
          m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));

          // Add local broadcast record to the routing table
          Ptr<NetDevice> dev = m_ipv4->GetNetDevice (
              m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
          RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true,
                                /*seqno=*/ 0, /*iface=*/ iface, /*hops=*/ 1,
                                /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
                                    ↪ Simulator::GetMaximumSimulationTime ());
          m_routingTable.AddRoute (rt);
        }
    }
  else
    {
      NS_LOG_LOGIC ("EA_AODV does not work with more then one address per each interface. Ignore added
          ↪ address");
    }
}

void
RoutingProtocol::NotifyRemoveAddress (uint32_t i, Ipv4InterfaceAddress address)
{
  NS_LOG_FUNCTION (this);
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (address);
  if (socket)
    {
      m_routingTable.DeleteAllRoutesFromInterface (address);
      socket->Close ();
      m_socketAddresses.erase (socket);

      Ptr<Socket> unicastSocket = FindSubnetBroadcastSocketWithInterfaceAddress (address);
      if (unicastSocket)
        {
          unicastSocket->Close ();
          m_socketAddresses.erase (unicastSocket);
        }

      Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
      if (l3->GetNAddresses (i))
```

```
930       {
931         Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
932         // Create a socket to listen only on this interface
933         Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
934                                                    UdpSocketFactory::GetTypeId ());
935         NS_ASSERT (socket != 0);
936         socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvEaAodv, this));
937         // Bind to any IP address so that broadcasts can be received
938         socket->BindToNetDevice (l3->GetNetDevice (i));
939         socket->Bind (InetSocketAddress (iface.GetLocal (), EA_AODV_PORT));
940         socket->SetAllowBroadcast (true);
941         socket->SetIpRecvTtl (true);
942         m_socketAddresses.insert (std::make_pair (socket, iface));
943 
944         // create also a unicast socket
945         socket = Socket::CreateSocket (GetObject<Node> (),
946                                        UdpSocketFactory::GetTypeId ());
947         NS_ASSERT (socket != 0);
948         socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvEaAodv, this));
949         socket->BindToNetDevice (l3->GetNetDevice (i));
950         socket->Bind (InetSocketAddress (iface.GetBroadcast (), EA_AODV_PORT));
951         socket->SetAllowBroadcast (true);
952         socket->SetIpRecvTtl (true);
953         m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));
954 
955         // Add local broadcast record to the routing table
956         Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
957         RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true,
              ↪ /*seqno=*/ 0, /*iface=*/ iface,
958                               /*hops=*/ 1, /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
                      ↪ Simulator::GetMaximumSimulationTime ());
959         m_routingTable.AddRoute (rt);
960       }
961     if (m_socketAddresses.empty ())
962       {
963         NS_LOG_LOGIC ("No␣aodv␣interfaces");
964         m_htimer.Cancel ();
965         m_nb.Clear ();
966         m_routingTable.Clear ();
967         return;
968       }
969   }
970  else
971   {
972     NS_LOG_LOGIC ("Remove␣address␣not␣participating␣in␣EA_AODV␣operation");
973   }
974 }
975 
976 bool
977 RoutingProtocol::IsMyOwnAddress (Ipv4Address src)
978 {
979  NS_LOG_FUNCTION (this << src);
980  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
981        m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
982   {
983     Ipv4InterfaceAddress iface = j->second;
984     if (src == iface.GetLocal ())
985       {
986         return true;
987       }
988   }
989  return false;
990 }
991 
992 Ptr<Ipv4Route>
993 RoutingProtocol::LoopbackRoute (const Ipv4Header & hdr, Ptr<NetDevice> oif) const
994 {
995  NS_LOG_FUNCTION (this << hdr);
```

```
  NS_ASSERT (m_lo != 0);
  Ptr<Ipv4Route> rt = Create<Ipv4Route> ();
  rt->SetDestination (hdr.GetDestination ());
  //
  // Source address selection here is tricky. The loopback route is
  // returned when AODV does not have a route; this causes the packet
  // to be looped back and handled (cached) in RouteInput() method
  // while a route is found. However, connection-oriented protocols
  // like TCP need to create an endpoint four-tuple (src, src port,
  // dst, dst port) and create a pseudo-header for checksumming. So,
  // AODV needs to guess correctly what the eventual source address
  // will be.
  //
  // For single interface, single address nodes, this is not a problem.
  // When there are possibly multiple outgoing interfaces, the policy
  // implemented here is to pick the first available AODV interface.
  // If RouteOutput() caller specified an outgoing interface, that
  // further constrains the selection of source address
  //
  std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j = m_socketAddresses.begin ();
  if (oif)
    {
      // Iterate to find an address on the oif device
      for (j = m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
        {
          Ipv4Address addr = j->second.GetLocal ();
          int32_t interface = m_ipv4->GetInterfaceForAddress (addr);
          if (oif == m_ipv4->GetNetDevice (static_cast<uint32_t> (interface)))
            {
              rt->SetSource (addr);
              break;
            }
        }
    }
  else
    {
      rt->SetSource (j->second.GetLocal ());
    }
  NS_ASSERT_MSG (rt->GetSource () != Ipv4Address (), "Valid␣AODV␣source␣address␣not␣found");
  rt->SetGateway (Ipv4Address ("127.0.0.1"));
  rt->SetOutputDevice (m_lo);
  return rt;
}

void
RoutingProtocol::SendRequest (Ipv4Address dst)
{
  if (m_energyLevel > m_aliveThreshold) {
    NS_LOG_FUNCTION ( this << dst);
      // A node SHOULD NOT originate more than RREQ_RATELIMIT RREQ messages per second.
      if (m_rreqCount == m_rreqRateLimit)
        {
          // EA-AODV to make decay functions work with idle time, make the time portion
          // of the Simulator::Schedule function a variable:
          // Simulator::Schedule (m_rreqRateLimitTimer.GetDelayLeft () + MicroSeconds (100),
          // &RoutingProtocol::SendRequest, this, dst);
          // into:
          // Time sendLatency = m_rreqRateLimitTimer.GetDelayLeft() + MicroSeconds(100);
          // energyLevel -= (sendLatency*idleDecay + sendDecay)
          // Simulator::Schedule (sendLatency, &RoutingProtocol::SendRequest,this,dest);

          Simulator::Schedule (m_rreqRateLimitTimer.GetDelayLeft () + MicroSeconds (100),
                               &RoutingProtocol::SendRequest, this, dst);
          return;
        }
      else
        {
          m_rreqCount++;
```

```
1064        }
1065      // Create RREQ header
1066      RreqHeader rreqHeader;
1067      rreqHeader.SetDst (dst);
1068
1069      RoutingTableEntry rt;
1070      // Using the Hop field in Routing Table to manage the expanding ring search
1071      uint16_t ttl = m_ttlStart;
1072      if (m_routingTable.LookupRoute (dst, rt))
1073        {
1074          if (rt.GetFlag () != IN_SEARCH)
1075            {
1076              ttl = std::min<uint16_t> (rt.GetHop () + m_ttlIncrement, m_netDiameter);
1077            }
1078          else
1079            {
1080              ttl = rt.GetHop () + m_ttlIncrement;
1081              if (ttl > m_ttlThreshold)
1082                {
1083                  ttl = m_netDiameter;
1084                }
1085            }
1086          if (ttl == m_netDiameter)
1087            {
1088              rt.IncrementRreqCnt ();
1089            }
1090          if (rt.GetValidSeqNo ())
1091            {
1092              rreqHeader.SetDstSeqno (rt.GetSeqNo ());
1093            }
1094          else
1095            {
1096              rreqHeader.SetUnknownSeqno (true);
1097            }
1098          rt.SetHop (ttl);
1099          rt.SetFlag (IN_SEARCH);
1100          rt.SetLifeTime (m_pathDiscoveryTime);
1101          m_routingTable.Update (rt);
1102        }
1103      else
1104        {
1105          rreqHeader.SetUnknownSeqno (true);
1106          Ptr<NetDevice> dev = 0;
1107          RoutingTableEntry newEntry (
1108            /*device=*/ dev, /*dst=*/ dst, /*validSeqNo=*/ false, /*seqno=*/ 0,
1109            /*iface=*/ Ipv4InterfaceAddress (),/*hop=*/ ttl, /*nextHop=*/ Ipv4Address (),
1110            /*lifeTime=*/ m_pathDiscoveryTime, 0, 0
1111          );
1112          // Check if TtlStart == NetDiameter
1113          if (ttl == m_netDiameter)
1114            {
1115              newEntry.IncrementRreqCnt ();
1116            }
1117          newEntry.SetFlag (IN_SEARCH);
1118          m_routingTable.AddRoute (newEntry);
1119        }
1120
1121      if (m_gratuitousReply)
1122        {
1123          rreqHeader.SetGratuitousRrep (true);
1124        }
1125      if (m_destinationOnly)
1126        {
1127          rreqHeader.SetDestinationOnly (true);
1128        }
1129
1130      m_seqNo++;
1131      rreqHeader.SetOriginSeqno (m_seqNo);
```

```
      m_requestId++;
      rreqHeader.SetId (m_requestId);

      // Send RREQ as subnet directed broadcast from each interface used by aodv
      for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
            m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
        {
          Ptr<Socket> socket = j->first;
          Ipv4InterfaceAddress iface = j->second;

          rreqHeader.SetOrigin (iface.GetLocal ());
          m_rreqIdCache.IsDuplicate (iface.GetLocal (), m_requestId);

          Ptr<Packet> packet = Create<Packet> ();
          SocketIpTtlTag tag;
          tag.SetTtl (ttl);
          packet->AddPacketTag (tag);
          packet->AddHeader (rreqHeader);
          TypeHeader tHeader (EA_AODVTYPE_RREQ);
          packet->AddHeader (tHeader);
          // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
          Ipv4Address destination;
          if (iface.GetMask () == Ipv4Mask::GetOnes ())
            {
              destination = Ipv4Address ("255.255.255.255");
            }
          else
            {
              destination = iface.GetBroadcast ();
            }
          //BUGOUT (ipv4Addr<<": Send RREQ with id " << rreqHeader.GetId () << ", X: "<<rreqHeader.GetLocX()
            //<<", Y: "<<rreqHeader.GetLocY()<<", and energy: "<<rreqHeader.GetEnergyAccum());
          NS_LOG_DEBUG ("Send␣RREQ␣with␣id␣" << rreqHeader.GetId () << "␣to␣socket");
          m_lastBcastTime = Simulator::Now ();
          // EA-AODV calculate time
          // NS3 Website
          // https://www.nsnam.org/doxygen/classns3_1_1_time.html
          Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (10, 20))),
               ↪ &RoutingProtocol::SendTo, this, socket, packet, destination);
        }
      ScheduleRreqRetry (dst);
    }
}

void
RoutingProtocol::SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address destination)
{
  // Energy Model
  RoutingTableEntry rt;
  m_routingTable.LookupRoute(destination,rt);
  BUGOUT(ipv4Addr<<":␣power␣decay␣for␣transmission␣during␣route␣discovery.");
  m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(rt),2);
  socket->SendTo (packet, 0, InetSocketAddress (destination, EA_AODV_PORT));

}
void
RoutingProtocol::ScheduleRreqRetry (Ipv4Address dst)
{
  NS_LOG_FUNCTION (this << dst);
  if (m_addressReqTimer.find (dst) == m_addressReqTimer.end ())
    {
      Timer timer (Timer::CANCEL_ON_DESTROY);
      m_addressReqTimer[dst] = timer;
    }
  m_addressReqTimer[dst].SetFunction (&RoutingProtocol::RouteRequestTimerExpire, this);
  m_addressReqTimer[dst].Remove ();
  m_addressReqTimer[dst].SetArguments (dst);
  RoutingTableEntry rt;
```

```
  m_routingTable.LookupRoute (dst, rt);
  Time retry;
  if (rt.GetHop () < m_netDiameter)
    {
      retry = 2 * m_nodeTraversalTime * (rt.GetHop () + m_timeoutBuffer);
    }
  else
    {
      NS_ABORT_MSG_UNLESS (rt.GetRreqCnt () > 0, "Unexpected value for GetRreqCount ()");
      uint16_t backoffFactor = rt.GetRreqCnt () - 1;
      NS_LOG_LOGIC ("Applying binary exponential backoff factor " << backoffFactor);
      retry = m_netTraversalTime * (1 << backoffFactor);
    }
  m_addressReqTimer[dst].Schedule (retry);
  NS_LOG_LOGIC ("Scheduled RREQ retry in " << retry.GetSeconds () << " seconds");
}

void
RoutingProtocol::RecvEaAodv (Ptr<Socket> socket)
{
  NS_LOG_FUNCTION (this << socket);
  if (m_energyLevel >= m_aliveThreshold) {
    //std::string cmd = "/NodeList/";
    //cmd += m_ipv4->GetObject<Node>()->GetId();
    //cmd += "/DeviceList/*/$ns3::RangePropagationLossModel::MaxRange";
    //Config::Set(cmd,DoubleValue(100.0));

    Address sourceAddress;
    Ptr<Packet> packet = socket->RecvFrom (sourceAddress);
    InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress);
    Ipv4Address sender = inetSourceAddr.GetIpv4 ();
    Ipv4Address receiver;

    if (m_socketAddresses.find (socket) != m_socketAddresses.end ())
      {
        receiver = m_socketAddresses[socket].GetLocal ();
      }
    else if (m_socketSubnetBroadcastAddresses.find (socket) != m_socketSubnetBroadcastAddresses.end ())
      {
        receiver = m_socketSubnetBroadcastAddresses[socket].GetLocal ();
      }
    else
      {
        NS_ASSERT_MSG (false, "Received a packet from an unknown socket");
      }
    NS_LOG_DEBUG ("EA_AODV node " << this << " received a EA_AODV packet from " << sender << " to " <<
        ↪ receiver);

    UpdateRouteToNeighbor (sender, receiver);
    TypeHeader tHeader (EA_AODVTYPE_RREQ);
    packet->RemoveHeader (tHeader);
    if (!tHeader.IsValid ())
      {
        NS_LOG_DEBUG ("EA_AODV message " << packet->GetUid () << " with unknown type received: " <<
            ↪ tHeader.Get () << ". Drop");
        return; // drop
      }
    switch (tHeader.Get ())
      {
      case EA_AODVTYPE_RREQ:
        {
          RecvRequest (packet, receiver, sender);
          break;
        }
      case EA_AODVTYPE_RREP:
        {
          RecvReply (packet, receiver, sender);
          break;
```

```
1265        }
1266      case EA_AODVTYPE_RERR:
1267        {
1268          RecvError (packet, sender);
1269          break;
1270        }
1271      case EA_AODVTYPE_RREP_ACK:
1272        {
1273          RecvReplyAck (sender);
1274          break;
1275        }
1276      }
1277  }
1278 }
1279
1280 bool
1281 RoutingProtocol::UpdateRouteLifeTime (Ipv4Address addr, Time lifetime)
1282 {
1283   NS_LOG_FUNCTION (this << addr << lifetime);
1284   RoutingTableEntry rt;
1285   if (m_routingTable.LookupRoute (addr, rt))
1286     {
1287       if (rt.GetFlag () == VALID)
1288         {
1289           NS_LOG_DEBUG ("Updating␣VALID␣route");
1290           rt.SetRreqCnt (0);
1291           rt.SetLifeTime (std::max (lifetime, rt.GetLifeTime ()));
1292           m_routingTable.Update (rt);
1293           return true;
1294         }
1295     }
1296   return false;
1297 }
1298
1299 void
1300 RoutingProtocol::UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver)
1301 {
1302   NS_LOG_FUNCTION (this << "sender␣" << sender << "␣receiver␣" << receiver);
1303   RoutingTableEntry toNeighbor;
1304   if (!m_routingTable.LookupRoute (sender, toNeighbor))
1305     {
1306       Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1307       RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ false, /*seqno=*/ 0,
1308                                   /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                          ↪ (receiver), 0),
1309                                   /*hops=*/ 1, /*next hop=*/ sender, /*lifetime=*/
                                          ↪ m_activeRouteTimeout);
1310       m_routingTable.AddRoute (newEntry);
1311     }
1312   else
1313     {
1314       Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1315       if (toNeighbor.GetValidSeqNo () && (toNeighbor.GetHop () == 1) && (toNeighbor.GetOutputDevice () == dev))
1316         {
1317           toNeighbor.SetLifeTime (std::max (m_activeRouteTimeout, toNeighbor.GetLifeTime ()));
1318         }
1319       else
1320         {
1321           RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ false, /*seqno=*/ 0,
1322                                       /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                              ↪ (receiver), 0),
1323                                       /*hops=*/ 1, /*next hop=*/ sender, /*lifetime=*/ std::max
                                              ↪ (m_activeRouteTimeout, toNeighbor.GetLifeTime ()));
1324           m_routingTable.Update (newEntry);
1325         }
1326     }
1327
1328 }
```

```
1329
1330 void
1331 RoutingProtocol::RecvRequest (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src)
1332 {
1333   NS_LOG_FUNCTION (this);
1334   RreqHeader rreqHeader;
1335   p->RemoveHeader (rreqHeader);
1336
1337   // A node ignores all RREQs received from any node in its blacklist
1338   RoutingTableEntry toPrev;
1339   if (m_routingTable.LookupRoute (src, toPrev))
1340     {
1341       if (toPrev.IsUnidirectional ())
1342         {
1343           NS_LOG_DEBUG ("Ignoring␣RREQ␣from␣node␣in␣blacklist");
1344           return;
1345         }
1346     }
1347
1348   uint32_t id = rreqHeader.GetId ();
1349   Ipv4Address origin = rreqHeader.GetOrigin ();
1350
1351   /*
1352    * Node checks to determine whether it has received a RREQ with the same Originator IP Address and RREQ ID.
1353    * If such a RREQ has been received, the node silently discards the newly received RREQ.
1354    */
1355   if (m_rreqIdCache.IsDuplicate (origin, id))
1356     {
1357       NS_LOG_DEBUG ("Ignoring␣RREQ␣due␣to␣duplicate");
1358       return;
1359     }
1360
1361   // Increment RREQ hop count
1362   uint8_t hop = rreqHeader.GetHopCount () + 1;
1363   rreqHeader.SetHopCount (hop);
1364
1365   /*
1366    * When the reverse route is created or updated, the following actions on the route are also carried out:
1367    * 1. the Originator Sequence Number from the RREQ is compared to the corresponding destination sequence
         ↪ number
1368    * in the route table entry and copied if greater than the existing value there
1369    * 2. the valid sequence number field is set to true;
1370    * 3. the next hop in the routing table becomes the node from which the RREQ was received
1371    * 4. the hop count is copied from the Hop Count in the RREQ message;
1372    * 5. the Lifetime is set to be the maximum of (ExistingLifetime, MinimalLifetime), where
1373    * MinimalLifetime = current time + 2*NetTraversalTime - 2*HopCount*NodeTraversalTime
1374    */
1375   RoutingTableEntry toOrigin;
1376   if (!m_routingTable.LookupRoute (origin, toOrigin))
1377     {
1378       Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1379       RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ origin, /*validSeno=*/ true, /*seqNo=*/
            ↪ rreqHeader.GetOriginSeqno (),
1380                                   /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                        ↪ (receiver), 0), /*hops=*/ hop,
1381                                   /*nextHop*/ src, /*timeLife=*/ Time ((2 * m_netTraversalTime - 2 *
                                        ↪ hop * m_nodeTraversalTime)), 0, 0);
1382       m_routingTable.AddRoute (newEntry);
1383     }
1384   else
1385     {
1386       if (toOrigin.GetValidSeqNo ())
1387         {
1388           if (int32_t (rreqHeader.GetOriginSeqno ()) - int32_t (toOrigin.GetSeqNo ()) > 0)
1389             {
1390               toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
1391             }
1392         }
```

```
    else
      {
        toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
      }
    toOrigin.SetValidSeqNo (true);
    toOrigin.SetNextHop (src);
    toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
    toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
    toOrigin.SetHop (hop);
    toOrigin.SetLifeTime (std::max (Time (2 * m_netTraversalTime - 2 * hop * m_nodeTraversalTime),
                                    toOrigin.GetLifeTime ()));
    m_routingTable.Update (toOrigin);
    //m_nb.Update (src, Time (AllowedHelloLoss * HelloInterval));
  }


RoutingTableEntry toNeighbor;
if (!m_routingTable.LookupRoute (src, toNeighbor))
  {
    NS_LOG_DEBUG ("Neighbor:" << src << " not found in routing table. Creating an entry");
    Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
    RoutingTableEntry newEntry (dev, src, false, rreqHeader.GetOriginSeqno (),
                                m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
                                1, src, m_activeRouteTimeout, 0, 0);
    m_routingTable.AddRoute (newEntry);
  }
else
  {
    toNeighbor.SetLifeTime (m_activeRouteTimeout);
    toNeighbor.SetValidSeqNo (false);
    toNeighbor.SetSeqNo (rreqHeader.GetOriginSeqno ());
    toNeighbor.SetFlag (VALID);
    toNeighbor.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
    toNeighbor.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
    toNeighbor.SetHop (1);
    toNeighbor.SetNextHop (src);
    m_routingTable.Update (toNeighbor);
  }
m_nb.Update (src, Time (m_allowedHelloLoss * m_helloInterval));

NS_LOG_LOGIC (receiver << " receive RREQ with hop count " << static_cast<uint32_t> (rreqHeader.GetHopCount
    ↪ ())
                       << " ID " << rreqHeader.GetId ()
                       << " to destination " << rreqHeader.GetDst ());

// A node generates a RREP if either:
// (i) it is itself the destination,
if (IsMyOwnAddress (rreqHeader.GetDst ()))
  {
    m_routingTable.LookupRoute (origin, toOrigin);
    NS_LOG_DEBUG ("Send reply since I am the destination");
    SendReply (rreqHeader, toOrigin);
    return;
  }
/*
 * (ii) or it has an active route to the destination, the destination sequence number in the node's
    ↪ existing route table entry for the destination
 * is valid and greater than or equal to the Destination Sequence Number of the RREQ, and the "destination
    ↪ only" flag is NOT set.
 */
RoutingTableEntry toDst;
Ipv4Address dst = rreqHeader.GetDst ();
if (m_routingTable.LookupRoute (dst, toDst))
  {
    /*
     * Drop RREQ, This node RREP will make a loop.
     */
    if (toDst.GetNextHop () == src)
```

```
1458        {
1459          NS_LOG_DEBUG ("Drop␣RREQ␣from␣" << src << ",␣dest␣next␣hop␣" << toDst.GetNextHop ());
1460          return;
1461        }
1462      /*
1463       * The Destination Sequence number for the requested destination is set to the maximum of the
              ↪ corresponding value
1464       * received in the RREQ message, and the destination sequence value currently maintained by the node for
              ↪ the requested destination.
1465       * However, the forwarding node MUST NOT modify its maintained value for the destination sequence
              ↪ number, even if the value
1466       * received in the incoming RREQ is larger than the value currently maintained by the forwarding node.
1467       */
1468      if ((rreqHeader.GetUnknownSeqno () || (int32_t (toDst.GetSeqNo ()) - int32_t (rreqHeader.GetDstSeqno ())
              ↪ >= 0))
1469          && toDst.GetValidSeqNo () )
1470        {
1471          if (!rreqHeader.GetDestinationOnly () && toDst.GetFlag () == VALID)
1472            {
1473              m_routingTable.LookupRoute (origin, toOrigin);
1474              SendReplyByIntermediateNode (toDst, toOrigin, rreqHeader.GetGratuitousRrep ());
1475              return;
1476            }
1477          rreqHeader.SetDstSeqno (toDst.GetSeqNo ());
1478          rreqHeader.SetUnknownSeqno (false);
1479        }
1480    }
1481
1482  SocketIpTtlTag tag;
1483  p->RemovePacketTag (tag);
1484  if (tag.GetTtl () < 2)
1485    {
1486      NS_LOG_DEBUG ("TTL␣exceeded.␣Drop␣RREQ␣origin␣" << src << "␣destination␣" << dst );
1487      return;
1488    }
1489
1490  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
1491        m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
1492    {
1493      Ptr<Socket> socket = j->first;
1494      Ipv4InterfaceAddress iface = j->second;
1495      Ptr<Packet> packet = Create<Packet> ();
1496      SocketIpTtlTag ttl;
1497      ttl.SetTtl (tag.GetTtl () - 1);
1498      packet->AddPacketTag (ttl);
1499      packet->AddHeader (rreqHeader);
1500      TypeHeader tHeader (EA_AODVTYPE_RREQ);
1501      packet->AddHeader (tHeader);
1502      // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
1503      Ipv4Address destination;
1504      if (iface.GetMask () == Ipv4Mask::GetOnes ())
1505        {
1506          destination = Ipv4Address ("255.255.255.255");
1507        }
1508      else
1509        {
1510          destination = iface.GetBroadcast ();
1511        }
1512      m_lastBcastTime = Simulator::Now ();
1513      Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
              ↪ &RoutingProtocol::SendTo, this, socket, packet, destination);
1514
1515    }
1516 }
1517
1518 void
1519 RoutingProtocol::SendReply (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin)
1520 {
```

```
1521   NS_LOG_FUNCTION (this << toOrigin.GetDestination ());
1522   /*
1523    * Destination node MUST increment its own sequence number by one if the sequence number in the RREQ packet
           ↪ is equal to that
1524    * incremented value. Otherwise, the destination does not change its sequence number before generating the
           ↪ RREP message.
1525    */
1526   if (!rreqHeader.GetUnknownSeqno () && (rreqHeader.GetDstSeqno () == m_seqNo + 1))
1527     {
1528       m_seqNo++;
1529     }
1530   // EA-AODV get node position
1531   this->getCoordinates();
1532   RrepHeader rrepHeader ( /*prefixSize=*/ 0, /*hops=*/ 0, /*dst=*/ rreqHeader.GetDst (),
1533                           /*dstSeqNo=*/ m_seqNo, /*origin=*/ toOrigin.GetDestination (),
1534                           /*lifeTime=*/ m_myRouteTimeout, nodeX, nodeY);
1535   Ptr<Packet> packet = Create<Packet> ();
1536   SocketIpTtlTag tag;
1537   tag.SetTtl (toOrigin.GetHop ());
1538   packet->AddPacketTag (tag);
1539   packet->AddHeader (rrepHeader);
1540   TypeHeader tHeader (EA_AODVTYPE_RREP);
1541   packet->AddHeader (tHeader);
1542   Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
1543   NS_ASSERT (socket);
1544   // Energy Model
1545     BUGOUT(ipv4Addr << ":␣Send␣RREP␣from␣X:␣" << rrepHeader.GetLocX() << ",␣Y:␣" << rrepHeader.GetLocY());
1546   BUGOUT(ipv4Addr<<":␣power␣decay␣for␣transmission␣during␣route␣discovery␣at␣time:"<<Simulator::Now()<<"␣to␣
           ↪ "<<toOrigin.GetNextHop());
1547   m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toOrigin),2);
1548   socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), EA_AODV_PORT));
1549 }
1550
1551 void
1552 RoutingProtocol::SendReplyByIntermediateNode (RoutingTableEntry & toDst, RoutingTableEntry & toOrigin, bool
       ↪ gratRep)
1553 {
1554   NS_LOG_FUNCTION (this);
1555   // EA-AODV
1556   RrepHeader rrepHeader (/*prefix size=*/ 0, /*hops=*/ toDst.GetHop (),
1557          /*dst=*/ toDst.GetDestination (), /*dst seqno=*/ toDst.GetSeqNo (),
1558          /*origin=*/ toOrigin.GetDestination (), /*lifetime=*/ toDst.GetLifeTime (),
1559          toOrigin.GetNextHopXLoc(), toOrigin.GetNextHopYLoc());
1560          // not sure if X and Y locations should be from origin or destination???
1561   /* If the node we received a RREQ for is a neighbor we are
1562    * probably facing a unidirectional link... Better request a RREP-ack
1563    */
1564   if (toDst.GetHop () == 1)
1565     {
1566       rrepHeader.SetAckRequired (true);
1567       RoutingTableEntry toNextHop;
1568       m_routingTable.LookupRoute (toOrigin.GetNextHop (), toNextHop);
1569       toNextHop.m_ackTimer.SetFunction (&RoutingProtocol::AckTimerExpire, this);
1570       toNextHop.m_ackTimer.SetArguments (toNextHop.GetDestination (), m_blackListTimeout);
1571       toNextHop.m_ackTimer.SetDelay (m_nextHopWait);
1572     }
1573   toDst.InsertPrecursor (toOrigin.GetNextHop ());
1574   toOrigin.InsertPrecursor (toDst.GetNextHop ());
1575   m_routingTable.Update (toDst);
1576   m_routingTable.Update (toOrigin);
1577
1578   Ptr<Packet> packet = Create<Packet> ();
1579   SocketIpTtlTag tag;
1580   tag.SetTtl (toOrigin.GetHop ());
1581   packet->AddPacketTag (tag);
1582   packet->AddHeader (rrepHeader);
1583   TypeHeader tHeader (EA_AODVTYPE_RREP);
1584   packet->AddHeader (tHeader);
```

```
1585   Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
1586   NS_ASSERT (socket);
1587   // Energy Model
1588   BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
1589   m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toOrigin),2);
1590   socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), EA_AODV_PORT));
1591
1592   // Generating gratuitous RREPs
1593   if (gratRep)
1594     {
1595       RrepHeader gratRepHeader (/*prefix size=*/ 0, /*hops=*/ toOrigin.GetHop (), /*dst=*/
            ↪ toOrigin.GetDestination (),
1596                                 /*dst seqno=*/ toOrigin.GetSeqNo (), /*origin=*/
                                      ↪ toDst.GetDestination (),
1597                                 /*lifetime=*/ toOrigin.GetLifeTime ());
1598       Ptr<Packet> packetToDst = Create<Packet> ();
1599       SocketIpTtlTag gratTag;
1600       gratTag.SetTtl (toDst.GetHop ());
1601       packetToDst->AddPacketTag (gratTag);
1602       packetToDst->AddHeader (gratRepHeader);
1603       TypeHeader type (EA_AODVTYPE_RREP);
1604       packetToDst->AddHeader (type);
1605       Ptr<Socket> socket = FindSocketWithInterfaceAddress (toDst.GetInterface ());
1606       NS_ASSERT (socket);
1607       NS_LOG_LOGIC ("Send gratuitous RREP " << packet->GetUid ());
1608       // Energy Model
1609       BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
1610       m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toDst),2);
1611       socket->SendTo (packetToDst, 0, InetSocketAddress (toDst.GetNextHop (), EA_AODV_PORT));
1612     }
1613 }
1614
1615 void
1616 RoutingProtocol::SendReplyAck (Ipv4Address neighbor)
1617 {
1618   NS_LOG_FUNCTION (this << " to " << neighbor);
1619   RrepAckHeader h;
1620   TypeHeader typeHeader (EA_AODVTYPE_RREP_ACK);
1621   Ptr<Packet> packet = Create<Packet> ();
1622   SocketIpTtlTag tag;
1623   tag.SetTtl (1);
1624   packet->AddPacketTag (tag);
1625   packet->AddHeader (h);
1626   packet->AddHeader (typeHeader);
1627   RoutingTableEntry toNeighbor;
1628   m_routingTable.LookupRoute (neighbor, toNeighbor);
1629   Ptr<Socket> socket = FindSocketWithInterfaceAddress (toNeighbor.GetInterface ());
1630   NS_ASSERT (socket);
1631   // Energy Model
1632   BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
1633   m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toNeighbor),2);
1634   socket->SendTo (packet, 0, InetSocketAddress (neighbor, EA_AODV_PORT));
1635 }
1636
1637 void
1638 RoutingProtocol::RecvReply (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address sender)
1639 {
1640   NS_LOG_FUNCTION (this << " src " << sender);
1641   RrepHeader rrepHeader;
1642   p->RemoveHeader (rrepHeader);
1643   Ipv4Address dst = rrepHeader.GetDst ();
1644   NS_LOG_LOGIC ("RREP destination " << dst << " RREP origin " << rrepHeader.GetOrigin ());
1645
1646   uint8_t hop = rrepHeader.GetHopCount () + 1;
1647   rrepHeader.SetHopCount (hop);
1648
1649   // If RREP is Hello message
1650   if (dst == rrepHeader.GetOrigin ())
```

```
1651    {
1652      ProcessHello (rrepHeader, receiver);
1653      return;
1654    }
1655
1656  /*
1657   * If the route table entry to the destination is created or updated, then the following actions occur:
1658   * - the route is marked as active,
1659   * - the destination sequence number is marked as valid,
1660   * - the next hop in the route entry is assigned to be the node from which the RREP is received,
1661   * which is indicated by the source IP address field in the IP header,
1662   * - the hop count is set to the value of the hop count from RREP message + 1
1663   * - the expiry time is set to the current time plus the value of the Lifetime in the RREP message,
1664   * - and the destination sequence number is the Destination Sequence Number in the RREP message.
1665   */
1666  Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1667  RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ dst, /*validSeqNo=*/ true, /*seqno=*/
        ↪ rrepHeader.GetDstSeqno (),
1668                                  /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                        ↪ (receiver), 0),/*hop=*/ hop,
1669                                  /*nextHop=*/ sender, /*lifeTime=*/ rrepHeader.GetLifeTime (),
1670                                  rrepHeader.GetLocX(), rrepHeader.GetLocY());
1671  RoutingTableEntry toDst;
1672  // EA-AODV get distance to node that received RREP from
1673  float rrepDistance = sqrt(pow(rrepHeader.GetLocX(),2) + pow(rrepHeader.GetLocY(),2));
1674  if (m_routingTable.LookupRoute (dst, toDst))
1675    {
1676      // get distance to next hop neighbor on current
1677      float entryDistance = sqrt(pow(toDst.GetNextHopXLoc(),2) + pow(toDst.GetNextHopYLoc(),2));
1678      /*
1679       * The existing entry is updated only in the following circumstances:
1680       * (i) the sequence number in the routing table is marked as invalid in route table entry.
1681       */
1682      if (!toDst.GetValidSeqNo ())
1683        {
1684          m_routingTable.Update (newEntry);
1685        }
1686      // (ii)the Destination Sequence Number in the RREP is greater than the node's copy of the destination
            ↪ sequence number and the known value is valid,
1687      else if ((int32_t (rrepHeader.GetDstSeqno ()) - int32_t (toDst.GetSeqNo ())) > 0)
1688        {
1689          m_routingTable.Update (newEntry);
1690        }
1691      else if ((rrepHeader.GetDstSeqno () == toDst.GetSeqNo ()))
1692        {
1693          // (iii) the sequence numbers are the same, but the route is marked as inactive.
1694          if (/*(rrepHeader.GetDstSeqno () == toDst.GetSeqNo ()) &&*/ (toDst.GetFlag () != VALID))
1695            {
1696              m_routingTable.Update (newEntry);
1697            }
1698
1699          // EA-AODV if distance to neighbor that sent RREP is smaller, choose that path
1700          else if (rrepDistance < entryDistance) {
1701            m_routingTable.Update(newEntry);
1702          }
1703          // if distance for current path is smaller and...
1704          // (iv) the sequence numbers are the same, and the New Hop Count is smaller than the hop count in
                ↪ route table entry.
1705          else if (/*(rrepHeader.GetDstSeqno () == toDst.GetSeqNo ()) &&*/ (hop < toDst.GetHop ()))
1706            {
1707              m_routingTable.Update (newEntry);
1708            }
1709        }
1710    }
1711  else
1712    {
1713      // The forward route for this destination is created if it does not already exist.
1714      NS_LOG_LOGIC ("add␣new␣route");
```

```
1715      m_routingTable.AddRoute (newEntry);
1716    }
1717  // Acknowledge receipt of the RREP by sending a RREP-ACK message back
1718  if (rrepHeader.GetAckRequired ())
1719    {
1720      SendReplyAck (sender);
1721      rrepHeader.SetAckRequired (false);
1722    }
1723  NS_LOG_LOGIC ("receiver " << receiver << " origin " << rrepHeader.GetOrigin ());
1724  if (IsMyOwnAddress (rrepHeader.GetOrigin ()))
1725    {
1726      if (toDst.GetFlag () == IN_SEARCH)
1727        {
1728          m_routingTable.Update (newEntry);
1729          m_addressReqTimer[dst].Remove ();
1730          m_addressReqTimer.erase (dst);
1731        }
1732      m_routingTable.LookupRoute (dst, toDst);
1733      SendPacketFromQueue (dst, toDst.GetRoute ());
1734      return;
1735    }
1736
1737  RoutingTableEntry toOrigin;
1738  if (!m_routingTable.LookupRoute (rrepHeader.GetOrigin (), toOrigin) || toOrigin.GetFlag () == IN_SEARCH)
1739    {
1740      return; // Impossible! drop.
1741    }
1742  toOrigin.SetLifeTime (std::max (m_activeRouteTimeout, toOrigin.GetLifeTime ()));
1743  m_routingTable.Update (toOrigin);
1744
1745  // Update information about precursors
1746  if (m_routingTable.LookupValidRoute (rrepHeader.GetDst (), toDst))
1747    {
1748      toDst.InsertPrecursor (toOrigin.GetNextHop ());
1749      m_routingTable.Update (toDst);
1750
1751      RoutingTableEntry toNextHopToDst;
1752      m_routingTable.LookupRoute (toDst.GetNextHop (), toNextHopToDst);
1753      toNextHopToDst.InsertPrecursor (toOrigin.GetNextHop ());
1754      m_routingTable.Update (toNextHopToDst);
1755
1756      toOrigin.InsertPrecursor (toDst.GetNextHop ());
1757      m_routingTable.Update (toOrigin);
1758
1759      RoutingTableEntry toNextHopToOrigin;
1760      m_routingTable.LookupRoute (toOrigin.GetNextHop (), toNextHopToOrigin);
1761      toNextHopToOrigin.InsertPrecursor (toDst.GetNextHop ());
1762      m_routingTable.Update (toNextHopToOrigin);
1763    }
1764  SocketIpTtlTag tag;
1765  p->RemovePacketTag (tag);
1766  if (tag.GetTtl () < 2)
1767    {
1768      NS_LOG_DEBUG ("TTL exceeded. Drop RREP destination " << dst << " origin " << rrepHeader.GetOrigin ());
1769      return;
1770    }
1771
1772  Ptr<Packet> packet = Create<Packet> ();
1773  SocketIpTtlTag ttl;
1774  ttl.SetTtl (tag.GetTtl () - 1);
1775  packet->AddPacketTag (ttl);
1776  packet->AddHeader (rrepHeader);
1777  TypeHeader tHeader (EA_AODVTYPE_RREP);
1778  packet->AddHeader (tHeader);
1779  Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
1780  NS_ASSERT (socket);
1781  // Energy Model
1782  BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
```

```
1783   m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toOrigin),2);
1784   socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), EA_AODV_PORT));
1785 }
1786
1787 void
1788 RoutingProtocol::RecvReplyAck (Ipv4Address neighbor)
1789 {
1790   NS_LOG_FUNCTION (this);
1791   RoutingTableEntry rt;
1792   if (m_routingTable.LookupRoute (neighbor, rt))
1793     {
1794       rt.m_ackTimer.Cancel ();
1795       rt.SetFlag (VALID);
1796       m_routingTable.Update (rt);
1797     }
1798 }
1799
1800 void
1801 RoutingProtocol::ProcessHello (RrepHeader const & rrepHeader, Ipv4Address receiver )
1802 {
1803   NS_LOG_FUNCTION (this << "from␣" << rrepHeader.GetDst ());
1804   /*
1805    * Whenever a node receives a Hello message from a neighbor, the node
1806    * SHOULD make sure that it has an active route to the neighbor, and
1807    * create one if necessary.
1808    */
1809   RoutingTableEntry toNeighbor;
1810   if (!m_routingTable.LookupRoute (rrepHeader.GetDst (), toNeighbor))
1811     {
1812       Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1813       RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ rrepHeader.GetDst (), /*validSeqNo=*/ true,
          ↪ /*seqno=*/ rrepHeader.GetDstSeqno (),
1814                                   /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                        ↪ (receiver), 0),
1815                                   /*hop=*/ 1, /*nextHop=*/ rrepHeader.GetDst (), /*lifeTime=*/
                                        ↪ rrepHeader.GetLifeTime ());
1816       m_routingTable.AddRoute (newEntry);
1817     }
1818   else
1819     {
1820       toNeighbor.SetLifeTime (std::max (Time (m_allowedHelloLoss * m_helloInterval), toNeighbor.GetLifeTime
          ↪ ()));
1821       toNeighbor.SetSeqNo (rrepHeader.GetDstSeqno ());
1822       toNeighbor.SetValidSeqNo (true);
1823       toNeighbor.SetFlag (VALID);
1824       toNeighbor.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
1825       toNeighbor.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
1826       toNeighbor.SetHop (1);
1827       toNeighbor.SetNextHop (rrepHeader.GetDst ());
1828       m_routingTable.Update (toNeighbor);
1829     }
1830   if (m_enableHello)
1831     {
1832       m_nb.Update (rrepHeader.GetDst (), Time (m_allowedHelloLoss * m_helloInterval));
1833     }
1834 }
1835
1836 void
1837 RoutingProtocol::RecvError (Ptr<Packet> p, Ipv4Address src )
1838 {
1839   NS_LOG_FUNCTION (this << "␣from␣" << src);
1840   RerrHeader rerrHeader;
1841   p->RemoveHeader (rerrHeader);
1842   std::map<Ipv4Address, uint32_t> dstWithNextHopSrc;
1843   std::map<Ipv4Address, uint32_t> unreachable;
1844   m_routingTable.GetListOfDestinationWithNextHop (src, dstWithNextHopSrc);
1845   std::pair<Ipv4Address, uint32_t> un;
1846   while (rerrHeader.RemoveUnDestination (un))
```

```
1847      {
1848        for (std::map<Ipv4Address, uint32_t>::const_iterator i =
1849              dstWithNextHopSrc.begin (); i != dstWithNextHopSrc.end (); ++i)
1850          {
1851            if (i->first == un.first)
1852              {
1853                unreachable.insert (un);
1854              }
1855          }
1856      }
1857
1858    std::vector<Ipv4Address> precursors;
1859    for (std::map<Ipv4Address, uint32_t>::const_iterator i = unreachable.begin ();
1860         i != unreachable.end (); )
1861      {
1862        if (!rerrHeader.AddUnDestination (i->first, i->second))
1863          {
1864            TypeHeader typeHeader (EA_AODVTYPE_RERR);
1865            Ptr<Packet> packet = Create<Packet> ();
1866            SocketIpTtlTag tag;
1867            tag.SetTtl (1);
1868            packet->AddPacketTag (tag);
1869            packet->AddHeader (rerrHeader);
1870            packet->AddHeader (typeHeader);
1871            SendRerrMessage (packet, precursors);
1872            rerrHeader.Clear ();
1873          }
1874        else
1875          {
1876            RoutingTableEntry toDst;
1877            m_routingTable.LookupRoute (i->first, toDst);
1878            toDst.GetPrecursors (precursors);
1879            ++i;
1880          }
1881      }
1882    if (rerrHeader.GetDestCount () != 0)
1883      {
1884        TypeHeader typeHeader (EA_AODVTYPE_RERR);
1885        Ptr<Packet> packet = Create<Packet> ();
1886        SocketIpTtlTag tag;
1887        tag.SetTtl (1);
1888        packet->AddPacketTag (tag);
1889        packet->AddHeader (rerrHeader);
1890        packet->AddHeader (typeHeader);
1891        SendRerrMessage (packet, precursors);
1892      }
1893    m_routingTable.InvalidateRoutesWithDst (unreachable);
1894  }
1895
1896  void
1897  RoutingProtocol::RouteRequestTimerExpire (Ipv4Address dst)
1898  {
1899    NS_LOG_LOGIC (this);
1900    RoutingTableEntry toDst;
1901    if (m_routingTable.LookupValidRoute (dst, toDst))
1902      {
1903        SendPacketFromQueue (dst, toDst.GetRoute ());
1904        NS_LOG_LOGIC ("route␣to␣" << dst << "␣found");
1905        return;
1906      }
1907    /*
1908     * If a route discovery has been attempted RreqRetries times at the maximum TTL without
1909     * receiving any RREP, all data packets destined for the corresponding destination SHOULD be
1910     * dropped from the buffer and a Destination Unreachable message SHOULD be delivered to the application.
1911     */
1912    if (toDst.GetRreqCnt () == m_rreqRetries)
1913      {
1914        NS_LOG_LOGIC ("route␣discovery␣to␣" << dst << "␣has␣been␣attempted␣RreqRetries␣(" << m_rreqRetries << ")␣
```

```
          ↪ times␣with␣ttl␣" << m_netDiameter);
1915      m_addressReqTimer.erase (dst);
1916      m_routingTable.DeleteRoute (dst);
1917      NS_LOG_DEBUG ("Route␣not␣found.␣Drop␣all␣packets␣with␣dst␣" << dst);
1918      m_queue.DropPacketWithDst (dst);
1919      if (isReachable == true) {
1920        SIMOUT("Node␣"<<ipv4Addr<<"␣unreachable␣at␣time␣"<<Simulator::Now().GetMilliSeconds()<<"ms.");
1921        isReachable = false;
1922      }
1923      return;
1924    }
1925
1926  if (toDst.GetFlag () == IN_SEARCH)
1927    {
1928      NS_LOG_LOGIC ("Resend␣RREQ␣to␣" << dst << "␣previous␣ttl␣" << toDst.GetHop ());
1929      SendRequest (dst);
1930    }
1931  else
1932    {
1933      NS_LOG_DEBUG ("Route␣down.␣Stop␣search.␣Drop␣packet␣with␣destination␣" << dst);
1934      m_addressReqTimer.erase (dst);
1935      m_routingTable.DeleteRoute (dst);
1936      m_queue.DropPacketWithDst (dst);
1937      if (isReachable == true) {
1938        SIMOUT("Node␣"<<ipv4Addr<<"␣unreachable␣at␣time␣"<<Simulator::Now().GetMilliSeconds()<<"ms.");
1939        isReachable = false;
1940      }
1941    }
1942  }
1943
1944  void
1945  RoutingProtocol::HelloTimerExpire ()
1946  {
1947    NS_LOG_FUNCTION (this);
1948    Time offset = Time (Seconds (0));
1949    if (m_lastBcastTime > Time (Seconds (0)))
1950      {
1951        offset = Simulator::Now () - m_lastBcastTime;
1952        NS_LOG_DEBUG ("Hello␣deferred␣due␣to␣last␣bcast␣at:" << m_lastBcastTime);
1953      }
1954    else
1955      {
1956        SendHello ();
1957      }
1958    m_htimer.Cancel ();
1959    Time diff = m_helloInterval - offset;
1960    m_htimer.Schedule (std::max (Time (Seconds (0)), diff));
1961    m_lastBcastTime = Time (Seconds (0));
1962  }
1963
1964  void
1965  RoutingProtocol::RreqRateLimitTimerExpire ()
1966  {
1967    NS_LOG_FUNCTION (this);
1968    m_rreqCount = 0;
1969    m_rreqRateLimitTimer.Schedule (Seconds (1));
1970  }
1971
1972  void
1973  RoutingProtocol::RerrRateLimitTimerExpire ()
1974  {
1975    NS_LOG_FUNCTION (this);
1976    m_rerrCount = 0;
1977    m_rerrRateLimitTimer.Schedule (Seconds (1));
1978  }
1979
1980  // EA_AODV used to decay energy level periodically based on idle time
1981  // and antenna receive power
```

```
1982 void RoutingProtocol::PowerDecayTimerExpire() {
1983   m_energyLevel -= (power_active+power_receive)*0.002;
1984   if (m_energyLevel < m_aliveThreshold) {
1985     Ipv4Address ipv4Addr = m_ipv4->GetAddress (1, 0).GetLocal ();
1986     getCoordinates();
1987     SIMOUT("Node␣"<<ipv4Addr<<"␣␣died␣at␣time␣"<<Simulator::Now().GetMilliSeconds()<<"ms.");
1988     m_powerDecayTimer.Cancel();
1989   } else {
1990     m_powerDecayTimer.Schedule(MilliSeconds(2));
1991   }
1992 }
1993
1994 void
1995 RoutingProtocol::AckTimerExpire (Ipv4Address neighbor, Time blacklistTimeout)
1996 {
1997   NS_LOG_FUNCTION (this);
1998   m_routingTable.MarkLinkAsUnidirectional (neighbor, blacklistTimeout);
1999 }
2000
2001 void
2002 RoutingProtocol::SendHello ()
2003 {
2004   NS_LOG_FUNCTION (this);
2005   /* Broadcast a RREP with TTL = 1 with the RREP message fields set as follows:
2006    * Destination IP Address The node's IP address.
2007    * Destination Sequence Number The node's latest sequence number.
2008    * Hop Count 0
2009    * Lifetime AllowedHelloLoss * HelloInterval
2010    */
2011   for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j = m_socketAddresses.begin (); j !=
         ↪ m_socketAddresses.end (); ++j)
2012     {
2013       Ptr<Socket> socket = j->first;
2014       Ipv4InterfaceAddress iface = j->second;
2015       RrepHeader helloHeader (/*prefix size=*/ 0, /*hops=*/ 0, /*dst=*/ iface.GetLocal (), /*dst seqno=*/
             ↪ m_seqNo,
2016                               /*origin=*/ iface.GetLocal (),/*lifetime=*/ Time
                                     ↪ (m_allowedHelloLoss * m_helloInterval));
2017       Ptr<Packet> packet = Create<Packet> ();
2018       SocketIpTtlTag tag;
2019       tag.SetTtl (1);
2020       packet->AddPacketTag (tag);
2021       packet->AddHeader (helloHeader);
2022       TypeHeader tHeader (EA_AODVTYPE_RREP);
2023       packet->AddHeader (tHeader);
2024       // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
2025       Ipv4Address destination;
2026       BUGOUT(ipv4Addr<<":␣sending␣Hello␣message␣to:␣"<<destination);
2027       if (iface.GetMask () == Ipv4Mask::GetOnes ())
2028         {
2029           destination = Ipv4Address ("255.255.255.255");
2030         }
2031       else
2032         {
2033           destination = iface.GetBroadcast ();
2034         }
2035       Time jitter = Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10)));
2036       Simulator::Schedule (jitter, &RoutingProtocol::SendTo, this, socket, packet, destination);
2037     }
2038 }
2039
2040 void
2041 RoutingProtocol::SendPacketFromQueue (Ipv4Address dst, Ptr<Ipv4Route> route)
2042 {
2043   NS_LOG_FUNCTION (this);
2044   QueueEntry queueEntry;
2045   while (m_queue.Dequeue (dst, queueEntry))
2046     {
```

```
      DeferredRouteOutputTag tag;
      Ptr<Packet> p = ConstCast<Packet> (queueEntry.GetPacket ());
      if (p->RemovePacketTag (tag)
          && tag.GetInterface () != -1
          && tag.GetInterface () != m_ipv4->GetInterfaceForDevice (route->GetOutputDevice ()))
        {
          NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
          return;
        }
      UnicastForwardCallback ucb = queueEntry.GetUnicastForwardCallback ();
      Ipv4Header header = queueEntry.GetIpv4Header ();
      header.SetSource (route->GetSource ());
      header.SetTtl (header.GetTtl () + 1); // compensate extra TTL decrement by fake loopback routing
      ucb (route, p, header);

      // Energy section
      this->getCoordinates();
      //int packet_size = PACKET_SIZE; //(Cameron): This needs to be adjusted when we have variable packet
          ↪ sizes. This works for now.
      m_lastUpdateTime = Simulator::Now();
      RoutingTableEntry rt;
      m_routingTable.LookupRoute(dst,rt);
      m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(rt),2);
    }
}

void
RoutingProtocol::SendRerrWhenBreaksLinkToNextHop (Ipv4Address nextHop)
{
  NS_LOG_FUNCTION (this << nextHop);
  if (m_energyLevel > m_aliveThreshold) {
    RerrHeader rerrHeader;
    std::vector<Ipv4Address> precursors;
    std::map<Ipv4Address, uint32_t> unreachable;

    RoutingTableEntry toNextHop;
    if (!m_routingTable.LookupRoute (nextHop, toNextHop))
      {
        return;
      }
    toNextHop.GetPrecursors (precursors);
    rerrHeader.AddUnDestination (nextHop, toNextHop.GetSeqNo ());
    m_routingTable.GetListOfDestinationWithNextHop (nextHop, unreachable);
    for (std::map<Ipv4Address, uint32_t>::const_iterator i = unreachable.begin (); i
         != unreachable.end (); )
      {
        if (!rerrHeader.AddUnDestination (i->first, i->second))
          {
            NS_LOG_LOGIC ("Send RERR message with maximum size.");
            TypeHeader typeHeader (EA_AODVTYPE_RERR);
            Ptr<Packet> packet = Create<Packet> ();
            SocketIpTtlTag tag;
            tag.SetTtl (1);
            packet->AddPacketTag (tag);
            packet->AddHeader (rerrHeader);
            packet->AddHeader (typeHeader);
            SendRerrMessage (packet, precursors);
            rerrHeader.Clear ();
          }
        else
          {
            RoutingTableEntry toDst;
            m_routingTable.LookupRoute (i->first, toDst);
            toDst.GetPrecursors (precursors);
            ++i;
          }
      }
    if (rerrHeader.GetDestCount () != 0)
```

```
    {
      TypeHeader typeHeader (EA_AODVTYPE_RERR);
      Ptr<Packet> packet = Create<Packet> ();
      SocketIpTtlTag tag;
      tag.SetTtl (1);
      packet->AddPacketTag (tag);
      packet->AddHeader (rerrHeader);
      packet->AddHeader (typeHeader);
      SendRerrMessage (packet, precursors);
    }
  unreachable.insert (std::make_pair (nextHop, toNextHop.GetSeqNo ()));
  m_routingTable.InvalidateRoutesWithDst (unreachable);
 }
}

void
RoutingProtocol::SendRerrWhenNoRouteToForward (Ipv4Address dst,
                                               uint32_t dstSeqNo, Ipv4Address origin)
{
 NS_LOG_FUNCTION (this);
 if (m_energyLevel > m_aliveThreshold) {
  // A node SHOULD NOT originate more than RERR_RATELIMIT RERR messages per second.
  if (m_rerrCount == m_rerrRateLimit)
    {
      // Just make sure that the RerrRateLimit timer is running and will expire
      NS_ASSERT (m_rerrRateLimitTimer.IsRunning ());
      // discard the packet and return
      NS_LOG_LOGIC ("RerrRateLimit␣reached␣at␣" << Simulator::Now ().GetSeconds () << "␣with␣timer␣delay␣
          ↪ left␣"
                                            << m_rerrRateLimitTimer.GetDelayLeft ().GetSeconds ()
                                            << ";␣suppressing␣RERR");
      return;
    }
  RerrHeader rerrHeader;
  rerrHeader.AddUnDestination (dst, dstSeqNo);
  RoutingTableEntry toOrigin;
  Ptr<Packet> packet = Create<Packet> ();
  SocketIpTtlTag tag;
  tag.SetTtl (1);
  packet->AddPacketTag (tag);
  packet->AddHeader (rerrHeader);
  packet->AddHeader (TypeHeader (EA_AODVTYPE_RERR));
  if (m_routingTable.LookupValidRoute (origin, toOrigin))
    {
      Ptr<Socket> socket = FindSocketWithInterfaceAddress (
          toOrigin.GetInterface ());
      NS_ASSERT (socket);
      NS_LOG_LOGIC ("Unicast␣RERR␣to␣the␣source␣of␣the␣data␣transmission");
      // Energy Model
      BUGOUT(ipv4Addr<<":␣power␣decay␣for␣transmission␣during␣route␣discovery.");
      m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toOrigin),2);
      socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), EA_AODV_PORT));
    }
  else
    {
      for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator i =
            m_socketAddresses.begin (); i != m_socketAddresses.end (); ++i)
        {
          Ptr<Socket> socket = i->first;
          Ipv4InterfaceAddress iface = i->second;
          NS_ASSERT (socket);
          NS_LOG_LOGIC ("Broadcast␣RERR␣message␣from␣interface␣" << iface.GetLocal ());
          // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
          Ipv4Address destination;
          if (iface.GetMask () == Ipv4Mask::GetOnes ())
            {
              destination = Ipv4Address ("255.255.255.255");
            }
```

```
          else
            {
              destination = iface.GetBroadcast ();
            }
          // Energy Model
          RoutingTableEntry rt;
          m_routingTable.LookupRoute(destination,rt);
          BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
          m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(rt),2);
          socket->SendTo (packet->Copy (), 0, InetSocketAddress (destination, EA_AODV_PORT));
        }
    }
  }
}

void
RoutingProtocol::SendRerrMessage (Ptr<Packet> packet, std::vector<Ipv4Address> precursors)
{
  NS_LOG_FUNCTION (this);
  if (m_energyLevel > m_aliveThreshold) {
    if (precursors.empty ())
      {
        NS_LOG_LOGIC ("No precursors");
        return;
      }
    // A node SHOULD NOT originate more than RERR_RATELIMIT RERR messages per second.
    if (m_rerrCount == m_rerrRateLimit)
      {
        // Just make sure that the RerrRateLimit timer is running and will expire
        NS_ASSERT (m_rerrRateLimitTimer.IsRunning ());
        // discard the packet and return
        NS_LOG_LOGIC ("RerrRateLimit reached at " << Simulator::Now ().GetSeconds () << " with timer delay
            ↪ left "
                                              << m_rerrRateLimitTimer.GetDelayLeft ().GetSeconds ()
                                              << "; suppressing RERR");
        return;
      }
    // If there is only one precursor, RERR SHOULD be unicast toward that precursor
    if (precursors.size () == 1)
      {
        RoutingTableEntry toPrecursor;
        if (m_routingTable.LookupValidRoute (precursors.front (), toPrecursor))
          {
            Ptr<Socket> socket = FindSocketWithInterfaceAddress (toPrecursor.GetInterface ());
            NS_ASSERT (socket);
            NS_LOG_LOGIC ("one precursor => unicast RERR to " << toPrecursor.GetDestination () << " from " <<
                ↪ toPrecursor.GetInterface ().GetLocal ());
            Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
                ↪ &RoutingProtocol::SendTo, this, socket, packet, precursors.front ());
            m_rerrCount++;
          }
        return;
      }

    // Should only transmit RERR on those interfaces which have precursor nodes for the broken route
    std::vector<Ipv4InterfaceAddress> ifaces;
    RoutingTableEntry toPrecursor;
    for (std::vector<Ipv4Address>::const_iterator i = precursors.begin (); i != precursors.end (); ++i)
      {
        if (m_routingTable.LookupValidRoute (*i, toPrecursor)
            && std::find (ifaces.begin (), ifaces.end (), toPrecursor.GetInterface ()) == ifaces.end ())
          {
            ifaces.push_back (toPrecursor.GetInterface ());
          }
      }

    for (std::vector<Ipv4InterfaceAddress>::const_iterator i = ifaces.begin (); i != ifaces.end (); ++i)
      {
```

```
2246        Ptr<Socket> socket = FindSocketWithInterfaceAddress (*i);
2247        NS_ASSERT (socket);
2248        NS_LOG_LOGIC ("Broadcast␣RERR␣message␣from␣interface␣" << i->GetLocal ());
2249        // std::cout << "Broadcast RERR message from interface " << i->GetLocal () << std::endl;
2250        // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
2251        Ptr<Packet> p = packet->Copy ();
2252        Ipv4Address destination;
2253        if (i->GetMask () == Ipv4Mask::GetOnes ())
2254          {
2255            destination = Ipv4Address ("255.255.255.255");
2256          }
2257        else
2258          {
2259            destination = i->GetBroadcast ();
2260          }
2261        Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
               ↪ &RoutingProtocol::SendTo, this, socket, p, destination);
2262      }
2263    }
2264  }
2265
2266  Ptr<Socket>
2267  RoutingProtocol::FindSocketWithInterfaceAddress (Ipv4InterfaceAddress addr ) const
2268  {
2269    NS_LOG_FUNCTION (this << addr);
2270    for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
2271          m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
2272      {
2273        Ptr<Socket> socket = j->first;
2274        Ipv4InterfaceAddress iface = j->second;
2275        if (iface == addr)
2276          {
2277            return socket;
2278          }
2279      }
2280    Ptr<Socket> socket;
2281    return socket;
2282  }
2283
2284  Ptr<Socket>
2285  RoutingProtocol::FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress addr ) const
2286  {
2287    NS_LOG_FUNCTION (this << addr);
2288    for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
2289          m_socketSubnetBroadcastAddresses.begin (); j != m_socketSubnetBroadcastAddresses.end (); ++j)
2290      {
2291        Ptr<Socket> socket = j->first;
2292        Ipv4InterfaceAddress iface = j->second;
2293        if (iface == addr)
2294          {
2295            return socket;
2296          }
2297      }
2298    Ptr<Socket> socket;
2299    return socket;
2300  }
2301
2302  void
2303  RoutingProtocol::DoInitialize (void)
2304  {
2305    NS_LOG_FUNCTION (this);
2306    uint32_t startTime;
2307    if (m_enableHello)
2308      {
2309        m_htimer.SetFunction (&RoutingProtocol::HelloTimerExpire, this);
2310        startTime = m_uniformRandomVariable->GetInteger (0, 100);
2311        NS_LOG_DEBUG ("Starting␣at␣time␣" << startTime << "ms");
2312        m_htimer.Schedule (MilliSeconds (startTime));
```

```
    }
  Ipv4RoutingProtocol::DoInitialize ();
}

void
RoutingProtocol::getCoordinates(void)
{
    Ptr<MobilityModel> mobNode = m_ipv4->GetObject<Node>()->GetObject<MobilityModel>();
    this->nodeX = mobNode->GetPosition().x;
    this->nodeY = mobNode->GetPosition().y;

    // trying this here to see if it works
    //std::string cmd = "/NodeList/";
    //cmd += m_ipv4->GetObject<Node>()->GetId();
    //cmd += "/DeviceList/*/$ns3::RangePropagationLossModel::MaxRange";
    //Config::Set(cmd,DoubleValue(0.0));
    return;
}

float
RoutingProtocol::GetDistanceBetweenNodes(RoutingTableEntry route)
{
    // Calculates hypotenuse from current and next node coordinates
    float a_sq = powf(abs(nodeX - route.GetNextHopXLoc()), 2);
    float b_sq = powf(abs(nodeY - route.GetNextHopYLoc()), 2);
    return sqrt(a_sq + b_sq);
}

} //namespace eaAodv
} //namespace ns3
```

LS-AODV

ls-aodv-routing-protocol.h

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * 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
 *
 * Based on
 * NS-2 AODV model developed by the CMU/MONARCH group and optimized and
 * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
 *
 * AODV-UU implementation by Erik Nordstrm of Uppsala University
 * http://core.it.uu.se/core/index.php/AODV-UU
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */
#ifndef LS_AODVROUTINGPROTOCOL_H
#define LS_AODVROUTINGPROTOCOL_H
```

```
30
31  // Cameron: This should change in the future when we allow variablity
32  #define PACKET_SIZE 4096
33  #define RREF_PACKET_SIZE 64
34  #define ESP32_VOLTAGE    3.3f
35
36  // Cameron: Burn rate for transmission per byte. Update this as determined.
37
38  #include "ls-aodv-rtable.h"
39  #include "ls-aodv-rqueue.h"
40  #include "ls-aodv-packet.h"
41  #include "ls-aodv-neighbor.h"
42  #include "ls-aodv-dpd.h"
43  #include "ns3/node.h"
44  #include "ns3/random-variable-stream.h"
45  #include "ns3/output-stream-wrapper.h"
46  #include "ns3/ipv4-routing-protocol.h"
47  #include "ns3/ipv4-interface.h"
48  #include "ns3/ipv4-l3-protocol.h"
49  #include <map>
50
51  namespace ns3 {
52  namespace lsaodv {
53  /**
54   * \ingroup lsaodv
55   *
56   * \brief AODV routing protocol
57   */
58  class RoutingProtocol : public Ipv4RoutingProtocol
59  {
60  public:
61    /**
62     * \brief Get the type ID.
63     * \return the object TypeId
64     */
65    static TypeId GetTypeId (void);
66    static const uint32_t LS_AODV_PORT;
67
68    // EA-AODV
69    static const float power_receive;
70    static const float power_active;
71    static const float m_aliveThreshold;
72    static const double TxConst;
73
74    /// constructor
75    RoutingProtocol ();
76    virtual ~RoutingProtocol ();
77    virtual void DoDispose ();
78
79    // Inherited from Ipv4RoutingProtocol
80    Ptr<Ipv4Route> RouteOutput (Ptr<Packet> p, const Ipv4Header &header, Ptr<NetDevice> oif,
        ↪ Socket::SocketErrno &sockerr);
81    bool RouteInput (Ptr<const Packet> p, const Ipv4Header &header, Ptr<const NetDevice> idev,
82                     UnicastForwardCallback ucb, MulticastForwardCallback mcb,
83                     LocalDeliverCallback lcb, ErrorCallback ecb);
84    virtual void NotifyInterfaceUp (uint32_t interface);
85    virtual void NotifyInterfaceDown (uint32_t interface);
86    virtual void NotifyAddAddress (uint32_t interface, Ipv4InterfaceAddress address);
87    virtual void NotifyRemoveAddress (uint32_t interface, Ipv4InterfaceAddress address);
88    virtual void SetIpv4 (Ptr<Ipv4> ipv4);
89    virtual void PrintRoutingTable (Ptr<OutputStreamWrapper> stream, Time::Unit unit = Time::S) const;
90
91    // Handle protocol parameters
92    /**
93     * Get maximum queue time
94     * \returns the maximum queue time
95     */
96    Time GetMaxQueueTime () const
```

```
97    {
98      return m_maxQueueTime;
99    }
100   /**
101    * Set the maximum queue time
102    * \param t the maximum queue time
103    */
104   void SetMaxQueueTime (Time t);
105   /**
106    * Get the maximum queue length
107    * \returns the maximum queue length
108    */
109   uint32_t GetMaxQueueLen () const
110   {
111     return m_maxQueueLen;
112   }
113   /**
114    * Set the maximum queue length
115    * \param len the maximum queue length
116    */
117   void SetMaxQueueLen (uint32_t len);
118   /**
119    * Get destination only flag
120    * \returns the destination only flag
121    */
122   bool GetDestinationOnlyFlag () const
123   {
124     return m_destinationOnly;
125   }
126   /**
127    * Set destination only flag
128    * \param f the destination only flag
129    */
130   void SetDestinationOnlyFlag (bool f)
131   {
132     m_destinationOnly = f;
133   }
134   /**
135    * Get gratuitous reply flag
136    * \returns the gratuitous reply flag
137    */
138   bool GetGratuitousReplyFlag () const
139   {
140     return m_gratuitousReply;
141   }
142   /**
143    * Set gratuitous reply flag
144    * \param f the gratuitous reply flag
145    */
146   void SetGratuitousReplyFlag (bool f)
147   {
148     m_gratuitousReply = f;
149   }
150   /**
151    * Set hello enable
152    * \param f the hello enable flag
153    */
154   void SetHelloEnable (bool f)
155   {
156     m_enableHello = f;
157   }
158   /**
159    * Get hello enable flag
160    * \returns the enable hello flag
161    */
162   bool GetHelloEnable () const
163   {
164     return m_enableHello;
```

```
165   }
166   /**
167    * Set broadcast enable flag
168    * \param f enable broadcast flag
169    */
170   void SetBroadcastEnable (bool f)
171   {
172     m_enableBroadcast = f;
173   }
174   /**
175    * Get broadcast enable flag
176    * \returns the broadcast enable flag
177    */
178   bool GetBroadcastEnable () const
179   {
180     return m_enableBroadcast;
181   }
182
183   /**
184    * Assign a fixed random variable stream number to the random variables
185    * used by this model. Return the number of streams (possibly zero) that
186    * have been assigned.
187    *
188    * \param stream first stream index to use
189    * \return the number of stream indices assigned by this model
190    */
191   int64_t AssignStreams (int64_t stream);
192
193 protected:
194   virtual void DoInitialize (void);
195 private:
196   // Protocol parameters.
197   uint32_t m_rreqRetries; ///< Maximum number of retransmissions of RREQ with TTL = NetDiameter to discover a
          ↪ route
198   uint16_t m_ttlStart; ///< Initial TTL value for RREQ.
199   uint16_t m_ttlIncrement; ///< TTL increment for each attempt using the expanding ring search for RREQ
          ↪ dissemination.
200   uint16_t m_ttlThreshold; ///< Maximum TTL value for expanding ring search, TTL = NetDiameter is used beyond
          ↪ this value.
201   uint16_t m_timeoutBuffer; ///< Provide a buffer for the timeout.
202   uint16_t m_rreqRateLimit; ///< Maximum number of RREQ per second.
203   uint16_t m_rerrRateLimit; ///< Maximum number of REER per second.
204   Time m_activeRouteTimeout; ///< Period of time during which the route is considered to be valid.
205   uint32_t m_netDiameter; ///< Net diameter measures the maximum possible number of hops between two nodes in
          ↪ the network
206   /**
207    * NodeTraversalTime is a conservative estimate of the average one hop traversal time for packets
208    * and should include queuing delays, interrupt processing times and transfer times.
209    */
210   Time m_nodeTraversalTime;
211   Time m_netTraversalTime; ///< Estimate of the average net traversal time.
212   Time m_pathDiscoveryTime; ///< Estimate of maximum time needed to find route in network.
213   Time m_myRouteTimeout; ///< Value of lifetime field in RREP generating by this node.
214   /**
215    * Every HelloInterval the node checks whether it has sent a broadcast within the last HelloInterval.
216    * If it has not, it MAY broadcast a Hello message
217    */
218   Time m_helloInterval;
219   uint32_t m_allowedHelloLoss; ///< Number of hello messages which may be loss for valid link
220   /**
221    * DeletePeriod is intended to provide an upper bound on the time for which an upstream node A
222    * can have a neighbor B as an active next hop for destination D, while B has invalidated the route to D.
223    */
224   Time m_deletePeriod;
225   Time m_nextHopWait; ///< Period of our waiting for the neighbour's RREP_ACK
226   Time m_blackListTimeout; ///< Time for which the node is put into the blacklist
227   uint32_t m_maxQueueLen; ///< The maximum number of packets that we allow a routing protocol to buffer.
```

```
228   Time m_maxQueueTime; ///< The maximum period of time that a routing protocol is allowed to buffer a packet
          ↪ for.
229   bool m_destinationOnly; ///< Indicates only the destination may respond to this RREQ.
230   bool m_gratuitousReply; ///< Indicates whether a gratuitous RREP should be unicast to the node originated
          ↪ route discovery.
231   bool m_enableHello; ///< Indicates whether a hello messages enable
232   bool m_enableBroadcast; ///< Indicates whether a a broadcast data packets forwarding enable
233   //\}
234
235   /// IP protocol
236   Ptr<Ipv4> m_ipv4;
237   /// Raw unicast socket per each IP interface, map socket -> iface address (IP + mask)
238   std::map< Ptr<Socket>, Ipv4InterfaceAddress > m_socketAddresses;
239   /// Raw subnet directed broadcast socket per each IP interface, map socket -> iface address (IP + mask)
240   std::map< Ptr<Socket>, Ipv4InterfaceAddress > m_socketSubnetBroadcastAddresses;
241   /// Loopback device used to defer RREQ until packet will be fully formed
242   Ptr<NetDevice> m_lo;
243
244   /// Routing table
245   RoutingTable m_routingTable;
246   /// A "drop-front" queue used by the routing layer to buffer packets to which it does not have a route.
247   RequestQueue m_queue;
248   /// Broadcast ID
249   uint32_t m_requestId;
250   /// Request sequence number
251   uint32_t m_seqNo;
252   /// Handle duplicated RREQ
253   IdCache m_rreqIdCache;
254   /// Handle duplicated broadcast/multicast packets
255   DuplicatePacketDetection m_dpd;
256   /// Handle neighbors
257   Neighbors m_nb;
258   /// Number of RREQs used for RREQ rate control
259   uint16_t m_rreqCount;
260   /// Number of RERRs used for RERR rate control
261   uint16_t m_rerrCount;
262   // EA-AODV
263   uint16_t m_rrepCount;
264 public:
265   // EA-AODV Instance Variables
266   double m_energyLevel;  // Remaining power level of instance node
267   double transDistance;  // Transmition distance of instance node
268   float nodeX;  // Instance node X coordinate
269   float nodeY; // Instance node Y coordinate
270   Time m_lastUpdateTime;
271   Ipv4Address ipv4Addr;
272   bool isReachable;
273 private:
274   /// Start protocol operation
275   void Start ();
276   /**
277    * Queue packet and send route request
278    *
279    * \param p the packet to route
280    * \param header the IP header
281    * \param ucb the UnicastForwardCallback function
282    * \param ecb the ErrorCallback function
283    */
284   void DeferredRouteOutput (Ptr<const Packet> p, const Ipv4Header & header, UnicastForwardCallback ucb,
          ↪ ErrorCallback ecb);
285   /**
286    * If route exists and is valid, forward packet.
287    *
288    * \param p the packet to route
289    * \param header the IP header
290    * \param ucb the UnicastForwardCallback function
291    * \param ecb the ErrorCallback function
292    * \returns true if forwarded
```

```
 */
bool Forwarding (Ptr<const Packet> p, const Ipv4Header & header, UnicastForwardCallback ucb, ErrorCallback
    ↪ ecb);
/**
 * Repeated attempts by a source node at route discovery for a single destination
 * use the expanding ring search technique.
 * \param dst the destination IP address
 */
void ScheduleRreqRetry (Ipv4Address dst);
/**
 * Set lifetime field in routing table entry to the maximum of existing lifetime and lt, if the entry exists
 * \param addr - destination address
 * \param lt - proposed time for lifetime field in routing table entry for destination with address addr.
 * \return true if route to destination address addr exist
 */
bool UpdateRouteLifeTime (Ipv4Address addr, Time lt);
/**
 * Update neighbor record.
 * \param receiver is supposed to be my interface
 * \param sender is supposed to be IP address of my neighbor.
 */
void UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver);
/**
 * Test whether the provided address is assigned to an interface on this node
 * \param src the source IP address
 * \returns true if the IP address is the node's IP address
 */
bool IsMyOwnAddress (Ipv4Address src);
/**
 * Find unicast socket with local interface address iface
 *
 * \param iface the interface
 * \returns the socket associated with the interface
 */
Ptr<Socket> FindSocketWithInterfaceAddress (Ipv4InterfaceAddress iface) const;
/**
 * Find subnet directed broadcast socket with local interface address iface
 *
 * \param iface the interface
 * \returns the socket associated with the interface
 */
Ptr<Socket> FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress iface) const;
/**
 * Process hello message
 *
 * \param rrepHeader RREP message header
 * \param receiverIfaceAddr receiver interface IP address
 */
void ProcessHello (RrepHeader const & rrepHeader, Ipv4Address receiverIfaceAddr);
/**
 * Create loopback route for given header
 *
 * \param header the IP header
 * \param oif the output interface net device
 * \returns the route
 */
Ptr<Ipv4Route> LoopbackRoute (const Ipv4Header & header, Ptr<NetDevice> oif) const;

///\name Receive control packets
//\{
/// Receive and process control packet
void Recvlsaodv (Ptr<Socket> socket);
/// Receive RREQ
void RecvRequest (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src);
/// Receive RREP
void RecvReply (Ptr<Packet> p, Ipv4Address my,Ipv4Address src);
/// Receive RREP_ACK
void RecvReplyAck (Ipv4Address neighbor);
```

```
360   /// Receive RERR from node with address src
361   void RecvError (Ptr<Packet> p, Ipv4Address src);
362   //\}
363
364   ///\name Send
365   //\{
366   /// Forward packet from route request queue
367   void SendPacketFromQueue (Ipv4Address dst, Ptr<Ipv4Route> route);
368   /// Send hello
369   void SendHello ();
370   /// Send RREQ
371   void SendRequest (Ipv4Address dst);
372   /// Send RREP
373   void SendReply (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin);
374   /** Send RREP by intermediate node
375    * \param toDst routing table entry to destination
376    * \param toOrigin routing table entry to originator
377    * \param gratRep indicates whether a gratuitous RREP should be unicast to destination
378    */
379   void SendReplyByIntermediateNode (RoutingTableEntry & toDst, RoutingTableEntry & toOrigin, bool gratRep);
380   /// Send RREP_ACK
381   void SendReplyAck (Ipv4Address neighbor);
382   /// Initiate RERR
383   void SendRerrWhenBreaksLinkToNextHop (Ipv4Address nextHop);
384   /// Forward RERR
385   void SendRerrMessage (Ptr<Packet> packet, std::vector<Ipv4Address> precursors);
386   /**
387    * Send RERR message when no route to forward input packet. Unicast if there is reverse route to
        ↪ originating node, broadcast otherwise.
388    * \param dst - destination node IP address
389    * \param dstSeqNo - destination node sequence number
390    * \param origin - originating node IP address
391    */
392   void SendRerrWhenNoRouteToForward (Ipv4Address dst, uint32_t dstSeqNo, Ipv4Address origin);
393   /// @}
394
395   /**
396    * Send packet to destination scoket
397    * \param socket - destination node socket
398    * \param packet - packet to send
399    * \param destination - destination node IP address
400    */
401   void SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address destination);
402
403   /// Hello timer
404   Timer m_htimer;
405   /// Schedule next send of hello message
406   void HelloTimerExpire ();
407   /// RREQ rate limit timer
408   Timer m_rreqRateLimitTimer;
409   /// Reset RREQ count and schedule RREQ rate limit timer with delay 1 sec.
410   void RreqRateLimitTimerExpire ();
411   /// RERR rate limit timer
412   Timer m_rerrRateLimitTimer;
413   /// Reset RERR count and schedule RERR rate limit timer with delay 1 sec.
414   void RerrRateLimitTimerExpire ();
415   /// Map IP address + RREQ timer.
416   std::map<Ipv4Address, Timer> m_addressReqTimer;
417
418   // EA-AODV begin
419   Timer m_powerDecayTimer;
420   void PowerDecayTimerExpire();
421   // EB-AODV
422   std::map<Ipv4Address, Timer> m_multipleRreqTimer;
423   void MultipleRREQRecvTimerExpire(Ipv4Address dst, RreqHeader rreqHeader,
424     RoutingTableEntry toOrigin);
425   // need to make this a map based on dst addresses
426   /**
```

```
   * Handle route discovery process
   * \param dst the destination IP address
   */
  void RouteRequestTimerExpire (Ipv4Address dst);
  /**
   * Mark link to neighbor node as unidirectional for blacklistTimeout
   *
   * \param neighbor the IP address of the neightbor node
   * \param blacklistTimeout the black list timeout time
   */
  void AckTimerExpire (Ipv4Address neighbor, Time blacklistTimeout);

  /// Provides uniform random variables.
  Ptr<UniformRandomVariable> m_uniformRandomVariable;
  /// Keep track of the last bcast time
  Time m_lastBcastTime;
  void getCoordinates(void);
  void DecayActiveEnergy(uint32_t oldValue, uint32_t newValue);
  float GetDistanceBetweenNodes(RoutingTableEntry route);
};

} //namespace lsaodv
} //namespace ns3

#endif /* LS_AODVROUTINGPROTOCOL_H */
```

ls-aodv-routing-protocol.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * 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
 *
 * Based on
 * NS-2 AODV model developed by the CMU/MONARCH group and optimized and
 * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
 *
 * AODV-UU implementation by Erik Nordstrm of Uppsala University
 * http://core.it.uu.se/core/index.php/AODV-UU
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */
#define NS_LOG_APPEND_CONTEXT \
  if (m_ipv4) { std::clog << "[node␣" << m_ipv4->GetObject<Node> ()->GetId () << "]␣"; }

#include "ls-aodv-routing-protocol.h"
#include "ns3/simulator.h"
#include "ns3/command-line.h"
#include "ns3/config.h"
#include "ns3/log.h"
#include "ns3/boolean.h"
#include "ns3/random-variable-stream.h"
#include "ns3/inet-socket-address.h"
```

```
#include "ns3/trace-source-accessor.h"
#include "ns3/udp-socket-factory.h"
#include "ns3/udp-l4-protocol.h"
#include "ns3/udp-header.h"
#include "ns3/wifi-net-device.h"
#include "ns3/adhoc-wifi-mac.h"
#include "ns3/mobility-model.h"
#include "ns3/string.h"
#include "ns3/pointer.h"
#include <algorithm>
#include <limits>
#include <cmath>

#define RREQ_WAIT_TIME 1

/* Dlsug and Simulation cout macros */
//#define DEBUG 1
#define SIMSTATS 1

#ifdef DEBUG
#define BUGOUT(x) do{std::cout<< x <<std::endl;}while(0)
#else
#define BUGOUT(x) do{}while(0)
#endif

#ifdef SIMSTATS
#define SIMOUT(x) do{std::cout<< x <<std::endl;}while(0)
#else
#define SIMOUT(x) do{}while(0)
#endif

// ENERGY MODEL CONSTANTS
#define ALPHA_TX    0.371354558f
#define MIN_RX_POW   1.58489319e-13f
#define LAMBDA2    0.015625f
#define PI2      9.869604401f
#define PATH_LOSS_EXP 3
namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("lsaodvRoutingProtocol");

namespace lsaodv {
NS_OBJECT_ENSURE_REGISTERED (RoutingProtocol);

/// UDP Port for LS_AODV control traffic
const uint32_t RoutingProtocol::LS_AODV_PORT = 654;

/**
 * EB-AODV Energy Model uses ESP32 Wifi Radio Characteristics for
 * IEEE 802.11b. Tranmission energy determined using Friis Propagation loss
 * equation to include transmission distance for potential energy savings.
 *
 **/
// Energy Model
const float RoutingProtocol::power_receive = 0.075 * ESP32_VOLTAGE;
const float RoutingProtocol::power_active = 0.0225 * ESP32_VOLTAGE;
// based on single-core @ 80 MHz with modem-sleep
const float RoutingProtocol::m_aliveThreshold = 0.001;
const double RoutingProtocol::TxConst=(double)(PACKET_SIZE*8)/1e6 * ESP32_VOLTAGE *
  MIN_RX_POW * 16 * PI2 / (ALPHA_TX * LAMBDA2);
/** determined by:
 * ALPHA_TX is proportional constant relating output antenna power to current
 * consumption: P_tx(W) = I_tx * ALPHA_TX
 * E_tx = P_tx * time
 * E_tx = #bits/(802.11b bitrate) * ESP32_VOLTAGE * I_tx
 * E_tx = #bits/(802.11b bitrate) * ESP32_VOLTAGE * MIN_RX_POW * 16*pi^2 * distance^2
 *  / (ALPHA_TX * LAMBDA^2)
 **/
```

```
/**
* \ingroup lsaodv
* \brief Tag used by AODV implementation
*/
class DeferredRouteOutputTag : public Tag
{

public:
  /**
   * \brief Constructor
   * \param o the output interface
   */
  DeferredRouteOutputTag (int32_t o = -1) : Tag (),
                                            m_oif (o)
  {
  }

  /**
   * \brief Get the type ID.
   * \return the object TypeId
   */
  static TypeId GetTypeId ()
  {
    static TypeId tid = TypeId ("ns3::lsaodv::DeferredRouteOutputTag")
      .SetParent<Tag> ()
      .SetGroupName ("lsaodv")
      .AddConstructor<DeferredRouteOutputTag> ()
    ;
    return tid;
  }

  TypeId GetInstanceTypeId () const
  {
    return GetTypeId ();
  }

  /**
   * \brief Get the output interface
   * \return the output interface
   */
  int32_t GetInterface () const
  {
    return m_oif;
  }

  /**
   * \brief Set the output interface
   * \param oif the output interface
   */
  void SetInterface (int32_t oif)
  {
    m_oif = oif;
  }

  uint32_t GetSerializedSize () const
  {
    return sizeof(int32_t);
  }

  void Serialize (TagBuffer i) const
  {
    i.WriteU32 (m_oif);
  }

  void Deserialize (TagBuffer i)
  {
    m_oif = i.ReadU32 ();
  }
```

```
175
176   void Print (std::ostream &os) const
177   {
178     os << "DeferredRouteOutputTag: output interface = " << m_oif;
179   }
180
181 private:
182   /// Positive if output device is fixed in RouteOutput
183   int32_t m_oif;
184 };
185
186 NS_OBJECT_ENSURE_REGISTERED (DeferredRouteOutputTag);
187
188
189 //-----------------------------------------------------------------------------
190 RoutingProtocol::RoutingProtocol ()
191   : m_rreqRetries (2),
192     m_ttlStart (1),
193     m_ttlIncrement (2),
194     m_ttlThreshold (7),
195     m_timeoutBuffer (2),
196     m_rreqRateLimit (10),
197     m_rerrRateLimit (10),
198     m_activeRouteTimeout (Seconds (3)),
199     m_netDiameter (35),
200     m_nodeTraversalTime (MilliSeconds (40)),
201     m_netTraversalTime (Time ((2 * m_netDiameter) * m_nodeTraversalTime)),
202     m_pathDiscoveryTime ( Time (2 * m_netTraversalTime)),
203     m_myRouteTimeout (Time (2 * std::max (m_pathDiscoveryTime, m_activeRouteTimeout))),
204     m_helloInterval (Seconds (1)),
205     m_allowedHelloLoss (2),
206     m_deletePeriod (Time (5 * std::max (m_activeRouteTimeout, m_helloInterval))),
207     m_nextHopWait (m_nodeTraversalTime + MilliSeconds (10)),
208     m_blackListTimeout (Time (m_rreqRetries * m_netTraversalTime)),
209     m_maxQueueLen (64),
210     m_maxQueueTime (Seconds (30)),
211     m_destinationOnly (false),
212     m_gratuitousReply (true),
213     m_enableHello (false),
214     m_routingTable (m_deletePeriod),
215     m_queue (m_maxQueueLen, m_maxQueueTime),
216     m_requestId (0),
217     m_seqNo (0),
218     m_rreqIdCache (m_pathDiscoveryTime),
219     m_dpd (m_pathDiscoveryTime),
220     m_nb (m_helloInterval),
221     m_rreqCount (0),
222     m_rerrCount (0),
223     m_rrepCount (0),
224     m_energyLevel(1000.0),
225     transDistance(250.0),
226     nodeX(-1),
227     nodeY(-1),
228     m_lastUpdateTime(Seconds(0.0)),
229     ipv4Addr(Ipv4Address::GetAny()),
230     isReachable(true),
231     m_htimer (Timer::CANCEL_ON_DESTROY),
232     m_rreqRateLimitTimer (Timer::CANCEL_ON_DESTROY),
233     m_rerrRateLimitTimer (Timer::CANCEL_ON_DESTROY),
234     m_powerDecayTimer(Timer::CANCEL_ON_DESTROY),
235     m_lastBcastTime (Seconds (0))
236 {
237   m_nb.SetCallback (MakeCallback (&RoutingProtocol::SendRerrWhenBreaksLinkToNextHop, this));
238 }
239
240 TypeId
241 RoutingProtocol::GetTypeId (void)
242 {
```

```
243   static TypeId tid = TypeId ("ns3::lsaodv::RoutingProtocol")
244     .SetParent<Ipv4RoutingProtocol> ()
245     .SetGroupName ("lsaodv")
246     .AddConstructor<RoutingProtocol> ()
247     .AddAttribute ("HelloInterval", "HELLO␣messages␣emission␣interval.",
248                    TimeValue (Seconds (1)),
249                    MakeTimeAccessor (&RoutingProtocol::m_helloInterval),
250                    MakeTimeChecker ())
251     .AddAttribute ("TtlStart", "Initial␣TTL␣value␣for␣RREQ.",
252                    UintegerValue (1),
253                    MakeUintegerAccessor (&RoutingProtocol::m_ttlStart),
254                    MakeUintegerChecker<uint16_t> ())
255     .AddAttribute ("TtlIncrement", "TTL␣increment␣for␣each␣attempt␣using␣the␣expanding␣ring␣search␣for␣RREQ␣
          ↪ dissemination.",
256                    UintegerValue (2),
257                    MakeUintegerAccessor (&RoutingProtocol::m_ttlIncrement),
258                    MakeUintegerChecker<uint16_t> ())
259     .AddAttribute ("TtlThreshold", "Maximum␣TTL␣value␣for␣expanding␣ring␣search,␣TTL␣=␣NetDiameter␣is␣used␣
          ↪ beyond␣this␣value.",
260                    UintegerValue (7),
261                    MakeUintegerAccessor (&RoutingProtocol::m_ttlThreshold),
262                    MakeUintegerChecker<uint16_t> ())
263     .AddAttribute ("TimeoutBuffer", "Provide␣a␣buffer␣for␣the␣timeout.",
264                    UintegerValue (2),
265                    MakeUintegerAccessor (&RoutingProtocol::m_timeoutBuffer),
266                    MakeUintegerChecker<uint16_t> ())
267     .AddAttribute ("RreqRetries", "Maximum␣number␣of␣retransmissions␣of␣RREQ␣to␣discover␣a␣route",
268                    UintegerValue (2),
269                    MakeUintegerAccessor (&RoutingProtocol::m_rreqRetries),
270                    MakeUintegerChecker<uint32_t> ())
271     .AddAttribute ("RreqRateLimit", "Maximum␣number␣of␣RREQ␣per␣second.",
272                    UintegerValue (10),
273                    MakeUintegerAccessor (&RoutingProtocol::m_rreqRateLimit),
274                    MakeUintegerChecker<uint32_t> ())
275     .AddAttribute ("RerrRateLimit", "Maximum␣number␣of␣RERR␣per␣second.",
276                    UintegerValue (10),
277                    MakeUintegerAccessor (&RoutingProtocol::m_rerrRateLimit),
278                    MakeUintegerChecker<uint32_t> ())
279     .AddAttribute ("NodeTraversalTime", "Conservative␣estimate␣of␣the␣average␣one␣hop␣traversal␣time␣for␣
          ↪ packets␣and␣should␣include␣"
280                    "queuing␣delays,␣interrupt␣processing␣times␣and␣transfer␣times.",
281                    TimeValue (MilliSeconds (40)),
282                    MakeTimeAccessor (&RoutingProtocol::m_nodeTraversalTime),
283                    MakeTimeChecker ())
284     .AddAttribute ("NextHopWait", "Period␣of␣our␣waiting␣for␣the␣neighbour's␣RREP_ACK␣=␣10␣ms␣+␣
          ↪ NodeTraversalTime",
285                    TimeValue (MilliSeconds (50)),
286                    MakeTimeAccessor (&RoutingProtocol::m_nextHopWait),
287                    MakeTimeChecker ())
288     .AddAttribute ("ActiveRouteTimeout", "Period␣of␣time␣during␣which␣the␣route␣is␣considered␣to␣be␣valid",
289                    TimeValue (Seconds (3)),
290                    MakeTimeAccessor (&RoutingProtocol::m_activeRouteTimeout),
291                    MakeTimeChecker ())
292     .AddAttribute ("MyRouteTimeout", "Value␣of␣lifetime␣field␣in␣RREP␣generating␣by␣this␣node␣=␣2␣*␣
          ↪ max(ActiveRouteTimeout,␣PathDiscoveryTime)",
293                    TimeValue (Seconds (11.2)),
294                    MakeTimeAccessor (&RoutingProtocol::m_myRouteTimeout),
295                    MakeTimeChecker ())
296     .AddAttribute ("BlackListTimeout", "Time␣for␣which␣the␣node␣is␣put␣into␣the␣blacklist␣=␣RreqRetries␣*␣
          ↪ NetTraversalTime",
297                    TimeValue (Seconds (5.6)),
298                    MakeTimeAccessor (&RoutingProtocol::m_blackListTimeout),
299                    MakeTimeChecker ())
300     .AddAttribute ("DeletePeriod", "DeletePeriod␣is␣intended␣to␣provide␣an␣upper␣bound␣on␣the␣time␣for␣which␣
          ↪ an␣upstream␣node␣A␣"
301                    "can␣have␣a␣neighbor␣B␣as␣an␣active␣next␣hop␣for␣destination␣D,␣while␣B␣has␣invalidated␣the␣
                         ↪ route␣to␣D."
302                    "␣=␣5␣*␣max␣(HelloInterval,␣ActiveRouteTimeout)",
```

```
                TimeValue (Seconds (15)),
                MakeTimeAccessor (&RoutingProtocol::m_deletePeriod),
                MakeTimeChecker ())
    .AddAttribute ("NetDiameter", "Net diameter measures the maximum possible number of hops between two
        ↪ nodes in the network",
                UintegerValue (35),
                MakeUintegerAccessor (&RoutingProtocol::m_netDiameter),
                MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("NetTraversalTime", "Estimate of the average net traversal time = 2 * NodeTraversalTime *
        ↪ NetDiameter",
                TimeValue (Seconds (2.8)),
                MakeTimeAccessor (&RoutingProtocol::m_netTraversalTime),
                MakeTimeChecker ())
    .AddAttribute ("PathDiscoveryTime", "Estimate of maximum time needed to find route in network = 2 *
        ↪ NetTraversalTime",
                TimeValue (Seconds (5.6)),
                MakeTimeAccessor (&RoutingProtocol::m_pathDiscoveryTime),
                MakeTimeChecker ())
    .AddAttribute ("MaxQueueLen", "Maximum number of packets that we allow a routing protocol to buffer.",
                UintegerValue (64),
                MakeUintegerAccessor (&RoutingProtocol::SetMaxQueueLen,
                                      &RoutingProtocol::GetMaxQueueLen),
                MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("MaxQueueTime", "Maximum time packets can be queued (in seconds)",
                TimeValue (Seconds (30)),
                MakeTimeAccessor (&RoutingProtocol::SetMaxQueueTime,
                                  &RoutingProtocol::GetMaxQueueTime),
                MakeTimeChecker ())
    .AddAttribute ("AllowedHelloLoss", "Number of hello messages which may be loss for valid link.",
                UintegerValue (2),
                MakeUintegerAccessor (&RoutingProtocol::m_allowedHelloLoss),
                MakeUintegerChecker<uint16_t> ())
    .AddAttribute ("GratuitousReply", "Indicates whether a gratuitous RREP should be unicast to the node
        ↪ originated route discovery.",
                BooleanValue (true),
                MakeBooleanAccessor (&RoutingProtocol::SetGratuitousReplyFlag,
                                     &RoutingProtocol::GetGratuitousReplyFlag),
                MakeBooleanChecker ())
    .AddAttribute ("DestinationOnly", "Indicates only the destination may respond to this RREQ.",
                BooleanValue (false),
                MakeBooleanAccessor (&RoutingProtocol::SetDestinationOnlyFlag,
                                     &RoutingProtocol::GetDestinationOnlyFlag),
                MakeBooleanChecker ())
    .AddAttribute ("EnableHello", "Indicates whether a hello messages enable.",
                BooleanValue (true),
                MakeBooleanAccessor (&RoutingProtocol::SetHelloEnable,
                                     &RoutingProtocol::GetHelloEnable),
                MakeBooleanChecker ())
    .AddAttribute ("EnableBroadcast", "Indicates whether a broadcast data packets forwarding enable.",
                BooleanValue (true),
                MakeBooleanAccessor (&RoutingProtocol::SetBroadcastEnable,
                                     &RoutingProtocol::GetBroadcastEnable),
                MakeBooleanChecker ())
    .AddAttribute ("UniformRv",
                "Access to the underlying UniformRandomVariable",
                StringValue ("ns3::UniformRandomVariable"),
                MakePointerAccessor (&RoutingProtocol::m_uniformRandomVariable),
                MakePointerChecker<UniformRandomVariable> ())
    .AddAttribute ("MaxPower",
                "Maximum battery power of a node",
                DoubleValue(100),
                MakeDoubleAccessor (&RoutingProtocol::m_energyLevel),
                MakeDoubleChecker<double> ())
    .AddAttribute ("TransDistance",
                "Transmission Distance for AODV algroithm",
                DoubleValue(250),
                MakeDoubleAccessor (&RoutingProtocol::transDistance),
                MakeDoubleChecker<double> ())
```

```
367   ;
368   return tid;
369 }
370
371 void
372 RoutingProtocol::SetMaxQueueLen (uint32_t len)
373 {
374   m_maxQueueLen = len;
375   m_queue.SetMaxQueueLen (len);
376 }
377 void
378 RoutingProtocol::SetMaxQueueTime (Time t)
379 {
380   m_maxQueueTime = t;
381   m_queue.SetQueueTimeout (t);
382 }
383
384 RoutingProtocol::~RoutingProtocol ()
385 {
386 }
387
388 void
389 RoutingProtocol::DoDispose ()
390 {
391   if (m_energyLevel > m_aliveThreshold) {
392     SIMOUT("Node␣"<<ipv4Addr<<"␣has␣"<<m_energyLevel<<"␣remaining␣power.");
393   }
394   m_ipv4 = 0;
395   for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::iterator iter =
396         m_socketAddresses.begin (); iter != m_socketAddresses.end (); iter++)
397     {
398       iter->first->Close ();
399     }
400   m_socketAddresses.clear ();
401   for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::iterator iter =
402         m_socketSubnetBroadcastAddresses.begin (); iter != m_socketSubnetBroadcastAddresses.end (); iter++)
403     {
404       iter->first->Close ();
405     }
406   m_socketSubnetBroadcastAddresses.clear ();
407   Ipv4RoutingProtocol::DoDispose ();
408 }
409
410 void
411 RoutingProtocol::PrintRoutingTable (Ptr<OutputStreamWrapper> stream, Time::Unit unit) const
412 {
413   *stream->GetStream () << "Node:␣" << m_ipv4->GetObject<Node> ()->GetId ()
414                         << ";␣Time:␣" << Now ().As (unit)
415                         << ",␣Local␣time:␣" << GetObject<Node> ()->GetLocalTime ().As (unit)
416                         << ",␣AODV␣Routing␣table" << std::endl;
417
418   m_routingTable.Print (stream);
419   *stream->GetStream () << std::endl;
420 }
421
422 int64_t
423 RoutingProtocol::AssignStreams (int64_t stream)
424 {
425   NS_LOG_FUNCTION (this << stream);
426   m_uniformRandomVariable->SetStream (stream);
427   return 1;
428 }
429
430 void
431 RoutingProtocol::Start ()
432 {
433   NS_LOG_FUNCTION (this);
434   if (m_enableHello)
```

```
435     {
436       m_nb.ScheduleTimer ();
437     }
438   m_rreqRateLimitTimer.SetFunction (&RoutingProtocol::RreqRateLimitTimerExpire,
439                                     this);
440   m_rreqRateLimitTimer.Schedule (Seconds (1));
441
442   m_rerrRateLimitTimer.SetFunction (&RoutingProtocol::RerrRateLimitTimerExpire,
443                                     this);
444   m_rerrRateLimitTimer.Schedule (Seconds (1));
445
446   // EA-AODV set timer for energy decay
447   m_powerDecayTimer.SetFunction (&RoutingProtocol::PowerDecayTimerExpire, this);
448   m_powerDecayTimer.Schedule(MilliSeconds(2));
449
450   ipv4Addr = m_ipv4->GetAddress (1, 0).GetLocal ();
451   getCoordinates();
452   SIMOUT("Node␣created␣"<<ipv4Addr<<"␣at␣pos:␣("<<nodeX<<","<<nodeY<<")␣with␣initial␣power␣"<<m_energyLevel);
453
454
455   //int nodeID = m_ipv4->GetObject<Node>()->GetId();
456   // connect application active energy calculation here
457   //Config::Connect("/NodeList/"+std::to_string(nodeID)+"/ApplicationList/*/$ns3::v4pingRand/ActiveTime",
458   // MakeCallback(&DecayActiveEnergy));
459 }
460
461 Ptr<Ipv4Route>
462 RoutingProtocol::RouteOutput (Ptr<Packet> p, const Ipv4Header &header,
463                               Ptr<NetDevice> oif, Socket::SocketErrno &sockerr)
464 {
465   NS_LOG_FUNCTION (this << header << (oif ? oif->GetIfIndex () : 0));
466   if (!p)
467     {
468       NS_LOG_DEBUG ("Packet␣is␣==␣0");
469       return LoopbackRoute (header, oif); // later
470     }
471   if (m_socketAddresses.empty ())
472     {
473       sockerr = Socket::ERROR_NOROUTETOHOST;
474       NS_LOG_LOGIC ("No␣aodv␣interfaces");
475       Ptr<Ipv4Route> route;
476       return route;
477     }
478   sockerr = Socket::ERROR_NOTERROR;
479   Ptr<Ipv4Route> route;
480   Ipv4Address dst = header.GetDestination ();
481   RoutingTableEntry rt;
482   if (m_routingTable.LookupValidRoute (dst, rt))
483     {
484     // EB-AODV valid route found, no route discovery needed, so decrement transmission power here!!!
485       route = rt.GetRoute ();
486       NS_ASSERT (route != 0);
487       NS_LOG_DEBUG ("Exist␣route␣to␣" << route->GetDestination () << "␣from␣interface␣" << route->GetSource
          ↪ ());
488       if (oif != 0 && route->GetOutputDevice () != oif)
489         {
490           NS_LOG_DEBUG ("Output␣device␣doesn't␣match.␣Dropped.");
491           sockerr = Socket::ERROR_NOROUTETOHOST;
492           return Ptr<Ipv4Route> ();
493         }
494       UpdateRouteLifeTime (dst, m_activeRouteTimeout);
495       UpdateRouteLifeTime (route->GetGateway (), m_activeRouteTimeout);
496       // EB-AODV POWER DECAY
497       BUGOUT(ipv4Addr<<":␣power␣decayed␣for␣transmission.");
498       double nextHopDistance = powf(GetDistanceBetweenNodes(rt),2);
499       m_energyLevel -= TxConst * powf(nextHopDistance,2);
500       return route;
501     }
```

```
502
503   // Valid route not found, in this case we return loopback.
504   // Actual route request will be deferred until packet will be fully formed,
505   // routed to loopback, received from loopback and passed to RouteInput (see below)
506   uint32_t iif = (oif ? m_ipv4->GetInterfaceForDevice (oif) : -1);
507   DeferredRouteOutputTag tag (iif);
508   NS_LOG_DEBUG ("Valid Route not found");
509   if (!p->PeekPacketTag (tag))
510     {
511       p->AddPacketTag (tag);
512     }
513   return LoopbackRoute (header, oif);
514 }
515
516 void
517 RoutingProtocol::DeferredRouteOutput (Ptr<const Packet> p, const Ipv4Header & header,
518                                       UnicastForwardCallback ucb, ErrorCallback ecb)
519 {
520   NS_LOG_FUNCTION (this << p << header);
521   NS_ASSERT (p != 0 && p != Ptr<Packet> ());
522   if (m_energyLevel > m_aliveThreshold) {
523     QueueEntry newEntry (p, header, ucb, ecb);
524     bool result = m_queue.Enqueue (newEntry);
525     if (result)
526       {
527         NS_LOG_LOGIC ("Add packet " << p->GetUid () << " to queue. Protocol " << (uint16_t) header.GetProtocol
                ↪ ());
528         RoutingTableEntry rt;
529         bool result = m_routingTable.LookupRoute (header.GetDestination (), rt);
530         if (!result || ((rt.GetFlag () != IN_SEARCH) && result))
531           {
532             NS_LOG_LOGIC ("Send new RREQ for outbound packet to " << header.GetDestination ());
533             SendRequest (header.GetDestination ());
534           }
535       }
536   }
537 }
538
539 bool
540 RoutingProtocol::RouteInput (Ptr<const Packet> p, const Ipv4Header &header,
541                              Ptr<const NetDevice> idev, UnicastForwardCallback ucb,
542                              MulticastForwardCallback mcb, LocalDeliverCallback lcb, ErrorCallback ecb)
543 {
544   NS_LOG_FUNCTION (this << p->GetUid () << header.GetDestination () << idev->GetAddress ());
545   if (m_energyLevel > m_aliveThreshold) {
546     if (m_socketAddresses.empty ())
547       {
548         NS_LOG_LOGIC ("No aodv interfaces");
549         return false;
550       }
551     NS_ASSERT (m_ipv4 != 0);
552     NS_ASSERT (p != 0);
553     // Check if input device supports IP
554     NS_ASSERT (m_ipv4->GetInterfaceForDevice (idev) >= 0);
555     int32_t iif = m_ipv4->GetInterfaceForDevice (idev);
556
557     Ipv4Address dst = header.GetDestination ();
558     Ipv4Address origin = header.GetSource ();
559
560     // Deferred route request
561     if (idev == m_lo)
562       {
563         DeferredRouteOutputTag tag;
564         if (p->PeekPacketTag (tag))
565           {
566             DeferredRouteOutput (p, header, ucb, ecb);
567             return true;
568           }
```

```
569       }
570
571     // Duplicate of own packet
572     if (IsMyOwnAddress (origin))
573       {
574         return true;
575       }
576
577     // AODV is not a multicast routing protocol
578     if (dst.IsMulticast ())
579       {
580         return false;
581       }
582
583     // Broadcast local delivery/forwarding
584     for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
585           m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
586       {
587         Ipv4InterfaceAddress iface = j->second;
588         if (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()) == iif)
589           {
590             if (dst == iface.GetBroadcast () || dst.IsBroadcast ())
591               {
592                 if (m_dpd.IsDuplicate (p, header))
593                   {
594                     NS_LOG_DEBUG ("Duplicated␣packet␣" << p->GetUid () << "␣from␣" << origin << ".␣Drop.");
595                     return true;
596                   }
597                 UpdateRouteLifeTime (origin, m_activeRouteTimeout);
598                 Ptr<Packet> packet = p->Copy ();
599                 if (lcb.IsNull () == false)
600                   {
601                     NS_LOG_LOGIC ("Broadcast␣local␣delivery␣to␣" << iface.GetLocal ());
602                     lcb (p, header, iif);
603                     // Fall through to additional processing
604                   }
605                 else
606                   {
607                     NS_LOG_ERROR ("Unable␣to␣deliver␣packet␣locally␣due␣to␣null␣callback␣" << p->GetUid () << "␣
                          ↪ from␣" << origin);
608                     ecb (p, header, Socket::ERROR_NOROUTETOHOST);
609                   }
610                 if (!m_enableBroadcast)
611                   {
612                     return true;
613                   }
614                 if (header.GetProtocol () == UdpL4Protocol::PROT_NUMBER)
615                   {
616                     UdpHeader udpHeader;
617                     p->PeekHeader (udpHeader);
618                     if (udpHeader.GetDestinationPort () == LS_AODV_PORT)
619                       {
620                         // AODV packets sent in broadcast are already managed
621                         return true;
622                       }
623                   }
624                 if (header.GetTtl () > 1)
625                   {
626                     NS_LOG_LOGIC ("Forward␣broadcast.␣TTL␣" << (uint16_t) header.GetTtl ());
627                     RoutingTableEntry toBroadcast;
628                     if (m_routingTable.LookupRoute (dst, toBroadcast))
629                       {
630                         Ptr<Ipv4Route> route = toBroadcast.GetRoute ();
631                         ucb (route, packet, header);
632                       }
633                     else
634                       {
635                         NS_LOG_DEBUG ("No␣route␣to␣forward␣broadcast.␣Drop␣packet␣" << p->GetUid ());
```

```
636                   }
637                 }
638               else
639                 {
640                   NS_LOG_DEBUG ("TTL␣exceeded.␣Drop␣packet␣" << p->GetUid ());
641                 }
642               return true;
643             }
644         }
645       }
646
647     // Unicast local delivery
648     if (m_ipv4->IsDestinationAddress (dst, iif))
649       {
650         UpdateRouteLifeTime (origin, m_activeRouteTimeout);
651         RoutingTableEntry toOrigin;
652         if (m_routingTable.LookupValidRoute (origin, toOrigin))
653           {
654             UpdateRouteLifeTime (toOrigin.GetNextHop (), m_activeRouteTimeout);
655             m_nb.Update (toOrigin.GetNextHop (), m_activeRouteTimeout);
656           }
657         if (lcb.IsNull () == false)
658           {
659             NS_LOG_LOGIC ("Unicast␣local␣delivery␣to␣" << dst);
660             lcb (p, header, iif);
661           }
662         else
663           {
664             NS_LOG_ERROR ("Unable␣to␣deliver␣packet␣locally␣due␣to␣null␣callback␣" << p->GetUid () << "␣from␣"
                  ↪ << origin);
665             ecb (p, header, Socket::ERROR_NOROUTETOHOST);
666           }
667         return true;
668       }
669
670     // Check if input device supports IP forwarding
671     if (m_ipv4->IsForwarding (iif) == false)
672       {
673         NS_LOG_LOGIC ("Forwarding␣disabled␣for␣this␣interface");
674         ecb (p, header, Socket::ERROR_NOROUTETOHOST);
675         return true;
676       }
677
678     // Forwarding
679     return Forwarding (p, header, ucb, ecb);
680   } else return false;
681 }
682
683 bool
684 RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
685                              UnicastForwardCallback ucb, ErrorCallback ecb)
686 {
687   NS_LOG_FUNCTION (this);
688   if (m_energyLevel > m_aliveThreshold) {
689     Ipv4Address dst = header.GetDestination ();
690     Ipv4Address origin = header.GetSource ();
691     m_routingTable.Purge ();
692     RoutingTableEntry toDst;
693     if (m_routingTable.LookupRoute (dst, toDst))
694       {
695         if (toDst.GetFlag () == VALID)
696           {
697             Ptr<Ipv4Route> route = toDst.GetRoute ();
698             NS_LOG_LOGIC (route->GetSource () << "␣forwarding␣to␣" << dst << "␣from␣" << origin << "␣packet␣"
                  ↪ << p->GetUid ());
699
700             /*
701              * Each time a route is used to forward a data packet, its Active Route
```

```
           * Lifetime field of the source, destination and the next hop on the
           * path to the destination is updated to be no less than the current
           * time plus ActiveRouteTimeout.
           */
          UpdateRouteLifeTime (origin, m_activeRouteTimeout);
          UpdateRouteLifeTime (dst, m_activeRouteTimeout);
          UpdateRouteLifeTime (route->GetGateway (), m_activeRouteTimeout);
          /*
           * Since the route between each originator and destination pair is expected to be symmetric, the
           * Active Route Lifetime for the previous hop, along the reverse path back to the IP source, is
               ↪ also updated
           * to be no less than the current time plus ActiveRouteTimeout
           */
          RoutingTableEntry toOrigin;
          m_routingTable.LookupRoute (origin, toOrigin);
          UpdateRouteLifeTime (toOrigin.GetNextHop (), m_activeRouteTimeout);

          m_nb.Update (route->GetGateway (), m_activeRouteTimeout);
          m_nb.Update (toOrigin.GetNextHop (), m_activeRouteTimeout);

          ucb (route, p, header);
          return true;
        }
       else
        {
          if (toDst.GetValidSeqNo ())
            {
              SendRerrWhenNoRouteToForward (dst, toDst.GetSeqNo (), origin);
              NS_LOG_DEBUG ("Drop␣packet␣" << p->GetUid () << "␣because␣no␣route␣to␣forward␣it.");
              return false;
            }
        }
     }
    NS_LOG_LOGIC ("route␣not␣found␣to␣" << dst << ".␣Send␣RERR␣message.");
    NS_LOG_DEBUG ("Drop␣packet␣" << p->GetUid () << "␣because␣no␣route␣to␣forward␣it.");
    SendRerrWhenNoRouteToForward (dst, 0, origin);
    return false;
  } else return false;
}

void
RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
{
  NS_ASSERT (ipv4 != 0);
  NS_ASSERT (m_ipv4 == 0);

  m_ipv4 = ipv4;

  // Create lo route. It is asserted that the only one interface up for now is loopback
  NS_ASSERT (m_ipv4->GetNInterfaces () == 1 && m_ipv4->GetAddress (0, 0).GetLocal () == Ipv4Address
      ↪ ("127.0.0.1"));
  m_lo = m_ipv4->GetNetDevice (0);
  NS_ASSERT (m_lo != 0);
  // Remember lo route
  RoutingTableEntry rt (/*device=*/ m_lo, /*dst=*/ Ipv4Address::GetLoopback (), /*know seqno=*/ true,
      ↪ /*seqno=*/ 0,
                                /*iface=*/ Ipv4InterfaceAddress (Ipv4Address::GetLoopback (), Ipv4Mask
                                    ↪ ("255.0.0.0")),
                                /*hops=*/ 1, /*next hop=*/ Ipv4Address::GetLoopback (),
                                /*lifetime=*/ Simulator::GetMaximumSimulationTime ());
  m_routingTable.AddRoute (rt);

  Simulator::ScheduleNow (&RoutingProtocol::Start, this);
}

void
RoutingProtocol::NotifyInterfaceUp (uint32_t i)
{
```

```
766   NS_LOG_FUNCTION (this << m_ipv4->GetAddress (i, 0).GetLocal ());
767   Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
768   if (l3->GetNAddresses (i) > 1)
769     {
770       NS_LOG_WARN ("LS_AODV does not work with more then one address per each interface.");
771     }
772   Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
773   if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
774     {
775       return;
776     }
777 
778   // Create a socket to listen only on this interface
779   Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
780                                              UdpSocketFactory::GetTypeId ());
781   NS_ASSERT (socket != 0);
782   socket->SetRecvCallback (MakeCallback (&RoutingProtocol::Recvlsaodv, this));
783   socket->BindToNetDevice (l3->GetNetDevice (i));
784   socket->Bind (InetSocketAddress (iface.GetLocal (), LS_AODV_PORT));
785   socket->SetAllowBroadcast (true);
786   socket->SetIpRecvTtl (true);
787   m_socketAddresses.insert (std::make_pair (socket, iface));
788 
789   // create also a subnet broadcast socket
790   socket = Socket::CreateSocket (GetObject<Node> (),
791                                  UdpSocketFactory::GetTypeId ());
792   NS_ASSERT (socket != 0);
793   socket->SetRecvCallback (MakeCallback (&RoutingProtocol::Recvlsaodv, this));
794   socket->BindToNetDevice (l3->GetNetDevice (i));
795   socket->Bind (InetSocketAddress (iface.GetBroadcast (), LS_AODV_PORT));
796   socket->SetAllowBroadcast (true);
797   socket->SetIpRecvTtl (true);
798   m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));
799 
800   // Add local broadcast record to the routing table
801   Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
802   RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true, /*seqno=*/ 0,
      ↪ /*iface=*/ iface,
803                                 /*hops=*/ 1, /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
                                      ↪ Simulator::GetMaximumSimulationTime ());
804   m_routingTable.AddRoute (rt);
805 
806   if (l3->GetInterface (i)->GetArpCache ())
807     {
808       m_nb.AddArpCache (l3->GetInterface (i)->GetArpCache ());
809     }
810 
811   // Allow neighbor manager use this interface for layer 2 feedback if possible
812   Ptr<WifiNetDevice> wifi = dev->GetObject<WifiNetDevice> ();
813   if (wifi == 0)
814     {
815       return;
816     }
817   Ptr<WifiMac> mac = wifi->GetMac ();
818   if (mac == 0)
819     {
820       return;
821     }
822 
823   mac->TraceConnectWithoutContext ("TxErrHeader", m_nb.GetTxErrorCallback ());
824 }
825 
826 void
827 RoutingProtocol::NotifyInterfaceDown (uint32_t i)
828 {
829   NS_LOG_FUNCTION (this << m_ipv4->GetAddress (i, 0).GetLocal ());
830 
831   // Disable layer 2 link state monitoring (if possible)
```

```
  Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
  Ptr<NetDevice> dev = l3->GetNetDevice (i);
  Ptr<WifiNetDevice> wifi = dev->GetObject<WifiNetDevice> ();
  if (wifi != 0)
    {
      Ptr<WifiMac> mac = wifi->GetMac ()->GetObject<AdhocWifiMac> ();
      if (mac != 0)
        {
          mac->TraceDisconnectWithoutContext ("TxErrHeader",
                                              m_nb.GetTxErrorCallback ());
          m_nb.DelArpCache (l3->GetInterface (i)->GetArpCache ());
        }
    }

  // Close socket
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
  NS_ASSERT (socket);
  socket->Close ();
  m_socketAddresses.erase (socket);

  // Close socket
  socket = FindSubnetBroadcastSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
  NS_ASSERT (socket);
  socket->Close ();
  m_socketSubnetBroadcastAddresses.erase (socket);

  if (m_socketAddresses.empty ())
    {
      NS_LOG_LOGIC ("No␣aodv␣interfaces");
      m_htimer.Cancel ();
      m_nb.Clear ();
      m_routingTable.Clear ();
      return;
    }
  m_routingTable.DeleteAllRoutesFromInterface (m_ipv4->GetAddress (i, 0));
}

void
RoutingProtocol::NotifyAddAddress (uint32_t i, Ipv4InterfaceAddress address)
{
  NS_LOG_FUNCTION (this << "␣interface␣" << i << "␣address␣" << address);
  Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
  if (!l3->IsUp (i))
    {
      return;
    }
  if (l3->GetNAddresses (i) == 1)
    {
      Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
      Ptr<Socket> socket = FindSocketWithInterfaceAddress (iface);
      if (!socket)
        {
          if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
            {
              return;
            }
          // Create a socket to listen only on this interface
          Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
                                                     UdpSocketFactory::GetTypeId ());
          NS_ASSERT (socket != 0);
          socket->SetRecvCallback (MakeCallback (&RoutingProtocol::Recvlsaodv,this));
          socket->BindToNetDevice (l3->GetNetDevice (i));
          socket->Bind (InetSocketAddress (iface.GetLocal (), LS_AODV_PORT));
          socket->SetAllowBroadcast (true);
          m_socketAddresses.insert (std::make_pair (socket, iface));

          // create also a subnet directed broadcast socket
          socket = Socket::CreateSocket (GetObject<Node> (),
```

```
900                                     UdpSocketFactory::GetTypeId ());
901           NS_ASSERT (socket != 0);
902           socket->SetRecvCallback (MakeCallback (&RoutingProtocol::Recvlsaodv, this));
903           socket->BindToNetDevice (l3->GetNetDevice (i));
904           socket->Bind (InetSocketAddress (iface.GetBroadcast (), LS_AODV_PORT));
905           socket->SetAllowBroadcast (true);
906           socket->SetIpRecvTtl (true);
907           m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));
908
909           // Add local broadcast record to the routing table
910           Ptr<NetDevice> dev = m_ipv4->GetNetDevice (
911               m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
912           RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true,
913                                 /*seqno=*/ 0, /*iface=*/ iface, /*hops=*/ 1,
914                                 /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
                                      ↪ Simulator::GetMaximumSimulationTime ());
915           m_routingTable.AddRoute (rt);
916         }
917       }
918     else
919       {
920         NS_LOG_LOGIC ("LS_AODV does not work with more then one address per each interface. Ignore added
                ↪ address");
921       }
922   }
923
924   void
925   RoutingProtocol::NotifyRemoveAddress (uint32_t i, Ipv4InterfaceAddress address)
926   {
927     NS_LOG_FUNCTION (this);
928     Ptr<Socket> socket = FindSocketWithInterfaceAddress (address);
929     if (socket)
930       {
931         m_routingTable.DeleteAllRoutesFromInterface (address);
932         socket->Close ();
933         m_socketAddresses.erase (socket);
934
935         Ptr<Socket> unicastSocket = FindSubnetBroadcastSocketWithInterfaceAddress (address);
936         if (unicastSocket)
937           {
938             unicastSocket->Close ();
939             m_socketAddresses.erase (unicastSocket);
940           }
941
942         Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
943         if (l3->GetNAddresses (i))
944           {
945             Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
946             // Create a socket to listen only on this interface
947             Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
948                                                        UdpSocketFactory::GetTypeId ());
949             NS_ASSERT (socket != 0);
950             socket->SetRecvCallback (MakeCallback (&RoutingProtocol::Recvlsaodv, this));
951             // Bind to any IP address so that broadcasts can be received
952             socket->BindToNetDevice (l3->GetNetDevice (i));
953             socket->Bind (InetSocketAddress (iface.GetLocal (), LS_AODV_PORT));
954             socket->SetAllowBroadcast (true);
955             socket->SetIpRecvTtl (true);
956             m_socketAddresses.insert (std::make_pair (socket, iface));
957
958             // create also a unicast socket
959             socket = Socket::CreateSocket (GetObject<Node> (),
960                                            UdpSocketFactory::GetTypeId ());
961             NS_ASSERT (socket != 0);
962             socket->SetRecvCallback (MakeCallback (&RoutingProtocol::Recvlsaodv, this));
963             socket->BindToNetDevice (l3->GetNetDevice (i));
964             socket->Bind (InetSocketAddress (iface.GetBroadcast (), LS_AODV_PORT));
965             socket->SetAllowBroadcast (true);
```

```
966           socket->SetIpRecvTtl (true);
967           m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));
968 
969           // Add local broadcast record to the routing table
970           Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
971           RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true,
                ↪ /*seqno=*/ 0, /*iface=*/ iface,
972                                 /*hops=*/ 1, /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
                     ↪ Simulator::GetMaximumSimulationTime ());
973           m_routingTable.AddRoute (rt);
974         }
975       if (m_socketAddresses.empty ())
976         {
977           NS_LOG_LOGIC ("No␣aodv␣interfaces");
978           m_htimer.Cancel ();
979           m_nb.Clear ();
980           m_routingTable.Clear ();
981           return;
982         }
983     }
984   else
985     {
986       NS_LOG_LOGIC ("Remove␣address␣not␣participating␣in␣LS_AODV␣operation");
987     }
988 }
989 
990 bool
991 RoutingProtocol::IsMyOwnAddress (Ipv4Address src)
992 {
993   NS_LOG_FUNCTION (this << src);
994   for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
995         m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
996     {
997       Ipv4InterfaceAddress iface = j->second;
998       if (src == iface.GetLocal ())
999         {
1000          return true;
1001        }
1002    }
1003  return false;
1004 }
1005 
1006 Ptr<Ipv4Route>
1007 RoutingProtocol::LoopbackRoute (const Ipv4Header & hdr, Ptr<NetDevice> oif) const
1008 {
1009   NS_LOG_FUNCTION (this << hdr);
1010   NS_ASSERT (m_lo != 0);
1011   Ptr<Ipv4Route> rt = Create<Ipv4Route> ();
1012   rt->SetDestination (hdr.GetDestination ());
1013   //
1014   // Source address selection here is tricky. The loopback route is
1015   // returned when AODV does not have a route; this causes the packet
1016   // to be looped back and handled (cached) in RouteInput() method
1017   // while a route is found. However, connection-oriented protocols
1018   // like TCP need to create an endpoint four-tuple (src, src port,
1019   // dst, dst port) and create a pseudo-header for checksumming. So,
1020   // AODV needs to guess correctly what the eventual source address
1021   // will be.
1022   //
1023   // For single interface, single address nodes, this is not a problem.
1024   // When there are possibly multiple outgoing interfaces, the policy
1025   // implemented here is to pick the first available AODV interface.
1026   // If RouteOutput() caller specified an outgoing interface, that
1027   // further constrains the selection of source address
1028   //
1029   std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j = m_socketAddresses.begin ();
1030   if (oif)
1031     {
```

```
1032      // Iterate to find an address on the oif device
1033      for (j = m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
1034        {
1035          Ipv4Address addr = j->second.GetLocal ();
1036          int32_t interface = m_ipv4->GetInterfaceForAddress (addr);
1037          if (oif == m_ipv4->GetNetDevice (static_cast<uint32_t> (interface)))
1038            {
1039              rt->SetSource (addr);
1040              break;
1041            }
1042        }
1043    }
1044  else
1045    {
1046      rt->SetSource (j->second.GetLocal ());
1047    }
1048  NS_ASSERT_MSG (rt->GetSource () != Ipv4Address (), "Valid␣AODV␣source␣address␣not␣found");
1049  rt->SetGateway (Ipv4Address ("127.0.0.1"));
1050  rt->SetOutputDevice (m_lo);
1051  return rt;
1052 }
1053
1054 void
1055 RoutingProtocol::SendRequest (Ipv4Address dst)
1056 {
1057  NS_LOG_FUNCTION ( this << dst);
1058  if (m_energyLevel > m_aliveThreshold) {
1059  // A node SHOULD NOT originate more than RREQ_RATELIMIT RREQ messages per second.
1060  if (m_rreqCount == m_rreqRateLimit)
1061    {
1062      // EA-AODV to make decay functions work with idle time, make the time portion
1063      // of the Simulator::Schedule function a variable:
1064      // Simulator::Schedule (m_rreqRateLimitTimer.GetDelayLeft () + MicroSeconds (100),
1065      // &RoutingProtocol::SendRequest, this, dst);
1066      // into:
1067      // Time sendLatency = m_rreqRateLimitTimer.GetDelayLeft() + MicroSeconds(100);
1068      // energyLevel -= (sendLatency*idleDecay + sendDecay)
1069      // Simulator::Schedule (sendLatency, &RoutingProtocol::SendRequest,this,dest);
1070
1071          Simulator::Schedule (m_rreqRateLimitTimer.GetDelayLeft () + MicroSeconds (100),
1072                               &RoutingProtocol::SendRequest, this, dst);
1073          return;
1074        }
1075      else
1076        {
1077          m_rreqCount++;
1078        }
1079      // Create RREQ header
1080      RreqHeader rreqHeader;
1081      rreqHeader.SetDst (dst);
1082
1083      RoutingTableEntry rt;
1084      // Using the Hop field in Routing Table to manage the expanding ring search
1085      uint16_t ttl = m_ttlStart;
1086      if (m_routingTable.LookupRoute (dst, rt))
1087        {
1088          if (rt.GetFlag () != IN_SEARCH)
1089            {
1090              ttl = std::min<uint16_t> (rt.GetHop () + m_ttlIncrement, m_netDiameter);
1091            }
1092          else
1093            {
1094              ttl = rt.GetHop () + m_ttlIncrement;
1095              if (ttl > m_ttlThreshold)
1096                {
1097                  ttl = m_netDiameter;
1098                }
1099            }
```

```
        if (ttl == m_netDiameter)
          {
            rt.IncrementRreqCnt ();
          }
        if (rt.GetValidSeqNo ())
          {
            rreqHeader.SetDstSeqno (rt.GetSeqNo ());
          }
        else
          {
            rreqHeader.SetUnknownSeqno (true);
          }
        rt.SetHop (ttl);
        rt.SetFlag (IN_SEARCH);
        rt.SetLifeTime (m_pathDiscoveryTime);
        m_routingTable.Update (rt);
      }
    else
      {
        rreqHeader.SetUnknownSeqno (true);
        Ptr<NetDevice> dev = 0;
        RoutingTableEntry newEntry (
          /*device=*/ dev, /*dst=*/ dst, /*validSeqNo=*/ false, /*seqno=*/ 0,
          /*iface=*/ Ipv4InterfaceAddress (),/*hop=*/ ttl, /*nextHop=*/ Ipv4Address (),
          /*lifeTime=*/ m_pathDiscoveryTime, 0, 0, 0
        );
        // Check if TtlStart == NetDiameter
        if (ttl == m_netDiameter)
          {
            newEntry.IncrementRreqCnt ();
          }
        newEntry.SetFlag (IN_SEARCH);
        m_routingTable.AddRoute (newEntry);
      }

    if (m_gratuitousReply)
      {
        rreqHeader.SetGratuitousRrep (true);
      }
    if (m_destinationOnly)
      {
        rreqHeader.SetDestinationOnly (true);
      }

    m_seqNo++;
    rreqHeader.SetOriginSeqno (m_seqNo);
    // EB-AODV add this node's information to new RREQ packet
    getCoordinates();
    rreqHeader.SetLocX(nodeX);
    rreqHeader.SetLocY(nodeY);
    rreqHeader.SetEnergyAccum(m_energyLevel);
    rreqHeader.SetNeighborEnergy(m_energyLevel);
    m_requestId++;
    rreqHeader.SetId (m_requestId);

    // Send RREQ as subnet directed broadcast from each interface used by aodv
    for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
           m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
      {
        Ptr<Socket> socket = j->first;
        Ipv4InterfaceAddress iface = j->second;

        rreqHeader.SetOrigin (iface.GetLocal ());
        m_rreqIdCache.IsDuplicate (iface.GetLocal (), m_requestId);

        Ptr<Packet> packet = Create<Packet> ();
        SocketIpTtlTag tag;
        tag.SetTtl (ttl);
```

```
1168          packet->AddPacketTag (tag);
1169          packet->AddHeader (rreqHeader);
1170          TypeHeader tHeader (LS_AODVTYPE_RREQ);
1171          packet->AddHeader (tHeader);
1172          // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
1173          Ipv4Address destination;
1174          if (iface.GetMask () == Ipv4Mask::GetOnes ())
1175            {
1176              destination = Ipv4Address ("255.255.255.255");
1177            }
1178          else
1179            {
1180              destination = iface.GetBroadcast ();
1181            }
1182          BUGOUT (ipv4Addr<<": Send RREQ with id " << rreqHeader.GetId () << ", X: "<<rreqHeader.GetLocX()
1183            <<", Y: "<<rreqHeader.GetLocY()<<", and energy: "<<rreqHeader.GetEnergyAccum());
1184          NS_LOG_DEBUG ("Send RREQ with id " << rreqHeader.GetId () << " to socket");
1185          m_lastBcastTime = Simulator::Now ();
1186          Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (10, 20))),
                ↪ &RoutingProtocol::SendTo, this, socket, packet, destination);
1187        }
1188      ScheduleRreqRetry (dst);
1189    }
1190  }
1191
1192  void
1193  RoutingProtocol::SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address destination)
1194  {
1195    // Energy Model
1196    RoutingTableEntry rt;
1197    m_routingTable.LookupRoute(destination,rt);
1198    BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
1199    m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(rt),PATH_LOSS_EXP);
1200    socket->SendTo (packet, 0, InetSocketAddress (destination, LS_AODV_PORT));
1201
1202  }
1203  void
1204  RoutingProtocol::ScheduleRreqRetry (Ipv4Address dst)
1205  {
1206    NS_LOG_FUNCTION (this << dst);
1207    if (m_addressReqTimer.find (dst) == m_addressReqTimer.end ())
1208      {
1209        Timer timer (Timer::CANCEL_ON_DESTROY);
1210        m_addressReqTimer[dst] = timer;
1211      }
1212    m_addressReqTimer[dst].SetFunction (&RoutingProtocol::RouteRequestTimerExpire, this);
1213    m_addressReqTimer[dst].Remove ();
1214    m_addressReqTimer[dst].SetArguments (dst);
1215    RoutingTableEntry rt;
1216    m_routingTable.LookupRoute (dst, rt);
1217    Time retry;
1218    if (rt.GetHop () < m_netDiameter)
1219      {
1220        retry = 2 * m_nodeTraversalTime * (rt.GetHop () + m_timeoutBuffer);
1221      }
1222    else
1223      {
1224        NS_ABORT_MSG_UNLESS (rt.GetRreqCnt () > 0, "Unexpected value for GetRreqCount ()");
1225        uint16_t backoffFactor = rt.GetRreqCnt () - 1;
1226        NS_LOG_LOGIC ("Applying binary exponential backoff factor " << backoffFactor);
1227        retry = m_netTraversalTime * (1 << backoffFactor);
1228      }
1229    m_addressReqTimer[dst].Schedule (retry);
1230    NS_LOG_LOGIC ("Scheduled RREQ retry in " << retry.GetSeconds () << " seconds");
1231  }
1232
1233  void
1234  RoutingProtocol::Recvlsaodv (Ptr<Socket> socket)
```

```
{
  NS_LOG_FUNCTION (this << socket);
  if (m_energyLevel >= m_aliveThreshold) {
    //std::string cmd = "/NodeList/";
    //cmd += m_ipv4->GetObject<Node>()->GetId();
    //cmd += "/DeviceList/*/$ns3::RangePropagationLossModel::MaxRange";
    //Config::Set(cmd,DoubleValue(100.0));

    Address sourceAddress;
    Ptr<Packet> packet = socket->RecvFrom (sourceAddress);
    InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress);
    Ipv4Address sender = inetSourceAddr.GetIpv4 ();
    Ipv4Address receiver;

    if (m_socketAddresses.find (socket) != m_socketAddresses.end ())
      {
        receiver = m_socketAddresses[socket].GetLocal ();
      }
    else if (m_socketSubnetBroadcastAddresses.find (socket) != m_socketSubnetBroadcastAddresses.end ())
      {
        receiver = m_socketSubnetBroadcastAddresses[socket].GetLocal ();
      }
    else
      {
        NS_ASSERT_MSG (false, "Received a packet from an unknown socket");
      }
    NS_LOG_DEBUG ("LS_AODV node " << this << " received a LS_AODV packet from " << sender << " to " <<
        ↪ receiver);

    UpdateRouteToNeighbor (sender, receiver);
    TypeHeader tHeader (LS_AODVTYPE_RREQ);
    packet->RemoveHeader (tHeader);
    if (!tHeader.IsValid ())
      {
        NS_LOG_DEBUG ("LS_AODV message " << packet->GetUid () << " with unknown type received: " <<
            ↪ tHeader.Get () << ". Drop");
        return; // drop
      }
    switch (tHeader.Get ())
      {
      case LS_AODVTYPE_RREQ:
        {
          RecvRequest (packet, receiver, sender);
          break;
        }
      case LS_AODVTYPE_RREP:
        {
          RecvReply (packet, receiver, sender);
          break;
        }
      case LS_AODVTYPE_RERR:
        {
          RecvError (packet, sender);
          break;
        }
      case LS_AODVTYPE_RREP_ACK:
        {
          RecvReplyAck (sender);
          break;
        }
      }
  }
}

bool
RoutingProtocol::UpdateRouteLifeTime (Ipv4Address addr, Time lifetime)
{
  NS_LOG_FUNCTION (this << addr << lifetime);
```

```
1301   RoutingTableEntry rt;
1302   if (m_routingTable.LookupRoute (addr, rt))
1303     {
1304       if (rt.GetFlag () == VALID)
1305         {
1306           NS_LOG_DEBUG ("Updating␣VALID␣route");
1307           rt.SetRreqCnt (0);
1308           rt.SetLifeTime (std::max (lifetime, rt.GetLifeTime ()));
1309           m_routingTable.Update (rt);
1310           return true;
1311         }
1312     }
1313   return false;
1314 }
1315
1316 void
1317 RoutingProtocol::UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver)
1318 {
1319   NS_LOG_FUNCTION (this << "sender␣" << sender << "␣receiver␣" << receiver);
1320   RoutingTableEntry toNeighbor;
1321   if (!m_routingTable.LookupRoute (sender, toNeighbor))
1322     {
1323       Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1324       RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ false, /*seqno=*/ 0,
1325                                   /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                          ↪ (receiver), 0),
1326                                   /*hops=*/ 1, /*next hop=*/ sender, /*lifetime=*/
                                          ↪ m_activeRouteTimeout);
1327       m_routingTable.AddRoute (newEntry);
1328     }
1329   else
1330     {
1331       Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1332       if (toNeighbor.GetValidSeqNo () && (toNeighbor.GetHop () == 1) && (toNeighbor.GetOutputDevice () == dev))
1333         {
1334           toNeighbor.SetLifeTime (std::max (m_activeRouteTimeout, toNeighbor.GetLifeTime ()));
1335         }
1336       else
1337         {
1338           RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ false, /*seqno=*/ 0,
1339                                       /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                              ↪ (receiver), 0),
1340                                       /*hops=*/ 1, /*next hop=*/ sender, /*lifetime=*/ std::max
                                              ↪ (m_activeRouteTimeout, toNeighbor.GetLifeTime ()));
1341           m_routingTable.Update (newEntry);
1342         }
1343     }
1344
1345 }
1346
1347 void
1348 RoutingProtocol::RecvRequest (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src)
1349 {
1350   NS_LOG_FUNCTION (this);
1351   RreqHeader rreqHeader;
1352   p->RemoveHeader (rreqHeader);
1353
1354   BUGOUT(ipv4Addr<<":␣receive␣RREQ␣with␣hop␣count␣" << static_cast<uint32_t> (rreqHeader.GetHopCount ())
1355     << "␣ID␣" << rreqHeader.GetId ()<<",␣seq.no.:␣"<<rreqHeader.GetOriginSeqno() << "␣from␣"<<src
1356     << "␣to␣" << rreqHeader.GetDst ()<<"␣with␣energyAccum:␣"<<rreqHeader.GetEnergyAccum()<<",␣at␣("
1357     <<rreqHeader.GetLocX()<<","<<rreqHeader.GetLocY()<<")."
1358   );
1359
1360   // EB-AODV
1361   bool updateRREQTimerValues = false;
1362   // A node ignores all RREQs received from any node in its blacklist
1363   RoutingTableEntry toPrev;
1364   if (m_routingTable.LookupRoute (src, toPrev))
```

```
1365    {
1366      if (toPrev.IsUnidirectional ())
1367        {
1368          NS_LOG_DEBUG ("Ignoring␣RREQ␣from␣node␣in␣blacklist");
1369          return;
1370        }
1371    }
1372
1373
1374  uint32_t id = rreqHeader.GetId ();
1375  Ipv4Address origin = rreqHeader.GetOrigin ();
1376
1377  if (IsMyOwnAddress (origin)) {
1378    NS_LOG_DEBUG("Received␣own␣RREQ,␣drop␣message");
1379    return;
1380  }
1381  /*if (m_rreqIdCache.IsDuplicate (origin, id) && rreqAvgEnergy <= rtEntryAvgEnergy)
1382    {
1383      BUGOUT(ipv4Addr<<": Ignoring RREQ because cached entry has same addr and ID");
1384      NS_LOG_DEBUG ("Ignoring RREQ due to duplicate");
1385      return;
1386    }
1387  */
1388
1389  // Increment RREQ hop count
1390  uint8_t hop = rreqHeader.GetHopCount () + 1;
1391  rreqHeader.SetHopCount (hop);
1392
1393  /*
1394   * When the reverse route is created or updated, the following actions on the route are also carried out:
1395   * 1. the Originator Sequence Number from the RREQ is compared to the corresponding destination sequence
       ↪ number
1396   * in the route table entry and copied if greater than the existing value there
1397   * 2. the valid sequence number field is set to true;
1398   * 3. the next hop in the routing table becomes the node from which the RREQ was received
1399   * 4. the hop count is copied from the Hop Count in the RREQ message;
1400   * 5. the Lifetime is set to be the maximum of (ExistingLifetime, MinimalLifetime), where
1401   * MinimalLifetime = current time + 2*NetTraversalTime - 2*HopCount*NodeTraversalTime
1402   */
1403  RoutingTableEntry toOrigin;
1404  if (!m_routingTable.LookupRoute (origin, toOrigin))
1405    {
1406      BUGOUT(ipv4Addr<<":␣Originator␣of␣RREQ␣not␣in␣Rtable,␣creating␣entry.");
1407      Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1408      RoutingTableEntry newEntry (
1409        /*device=*/ dev, /*dst=*/ origin, /*validSeno=*/ true,
1410        /*seqNo=*/ rreqHeader.GetOriginSeqno (),
1411        /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
1412        /*hops=*/ hop, /*nextHop*/ src,
1413        /*timeLife=*/ Time ((2 * m_netTraversalTime - 2 * hop * m_nodeTraversalTime)),
1414        // !!! EB-AODV added these, might be why original EA-AODV simulation didn't show good results???
1415        /*nextHopXLoc=*/ rreqHeader.GetLocX(), /*nextHopYLoc=*/rreqHeader.GetLocY(),
1416        /*energyAccum=*/rreqHeader.GetEnergyAccum()
1417      );
1418      m_routingTable.AddRoute (newEntry);
1419      updateRREQTimerValues = true;
1420    }
1421  // update current entry in routing table
1422  else
1423    {
1424      /*
1425       * Node checks to determine whether it has received a RREQ with the same Originator IP Address and RREQ
           ↪ ID.
1426       * If such a RREQ has been received, the node silently discards the newly received RREQ.
1427       */
1428      // EB-AODV calculate average energy per node along paths
1429      double rtEntryAvgEnergy = toOrigin.GetEnergyAccum()/((double)(toOrigin.GetHop()+1));
1430      double rreqAvgEnergy = rreqHeader.GetEnergyAccum()/((double)hop);
```

```
1431      BUGOUT(ipv4Addr<<": path energy for rtable entry: "<<rtEntryAvgEnergy<<" and RREQ: "<<rreqAvgEnergy);
1432
1433      // allows to ignore cached originators if the energy of the RREQ energy is less than
1434      // saved path or this is not the destination, creates distinct paths
1435      if (m_rreqIdCache.IsDuplicate (origin, id) &&
1436        (rreqAvgEnergy <= rtEntryAvgEnergy || !IsMyOwnAddress (rreqHeader.GetDst ()))
1437      )
1438        {
1439          BUGOUT(ipv4Addr<<": Ignoring RREQ because cached entry has same addr and ID");
1440          NS_LOG_DEBUG ("Ignoring RREQ due to duplicate");
1441          return;
1442        }
1443      else if (toOrigin.GetValidSeqNo ()) // valid seq.no. and not duplicate
1444        {
1445          // rreq seq.no. greater than rt entry, so update route
1446          if (int32_t (rreqHeader.GetOriginSeqno ()) - int32_t (toOrigin.GetSeqNo ()) > 0) {
1447              BUGOUT(ipv4Addr<<": RREQ seq.no. greater than stored seq.no. for the originator. Update
                      ↪ seq.no.");
1448              toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
1449              toOrigin.SetValidSeqNo (true);
1450              toOrigin.SetNextHop (src);
1451              toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
1452              toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
1453              toOrigin.SetHop (hop);
1454              toOrigin.SetLifeTime (std::max (Time (2 * m_netTraversalTime - 2 * hop * m_nodeTraversalTime),
1455                                              toOrigin.GetLifeTime ()));
1456              // EB-AODV
1457              toOrigin.SetNextHopXLoc(rreqHeader.GetLocX());
1458              toOrigin.SetNextHopYLoc(rreqHeader.GetLocY());
1459              toOrigin.SetEnergyAccum(rreqHeader.GetEnergyAccum());
1460              m_routingTable.Update (toOrigin);
1461              updateRREQTimerValues = true;
1462          // EB-AODV
1463          // average energy per node of rreq is greater than average energy per node of rt entry
1464          } else if (rreqAvgEnergy > rtEntryAvgEnergy) {
1465              BUGOUT(ipv4Addr<<": RREQ has higher path energy. Save this one with next hop:
                      ↪ "<<toOrigin.GetNextHop());
1466              toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
1467              toOrigin.SetValidSeqNo (true);
1468              toOrigin.SetNextHop (src);
1469              toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
1470              toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
1471              toOrigin.SetHop (hop);
1472              toOrigin.SetLifeTime (std::max (Time (2 * m_netTraversalTime - 2 * hop * m_nodeTraversalTime),
1473                                              toOrigin.GetLifeTime ()));
1474              // EB-AODV
1475              toOrigin.SetNextHopXLoc(rreqHeader.GetLocX());
1476              toOrigin.SetNextHopYLoc(rreqHeader.GetLocY());
1477              toOrigin.SetEnergyAccum(rreqHeader.GetEnergyAccum());
1478              m_routingTable.Update (toOrigin);
1479              updateRREQTimerValues = true;
1480            } else {
1481              BUGOUT(ipv4Addr<<": RREQ seq.no. not greater and path energy less than rt entry. Discard RREQ.");
1482            }
1483        }
1484      else // invalid seq.no. and not duplicate
1485        {
1486          BUGOUT(ipv4Addr<<": Stored seq.no. for originator expired. Update entry with RREQ");
1487          toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
1488          toOrigin.SetValidSeqNo (true);
1489          toOrigin.SetNextHop (src);
1490          toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
1491          toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
1492          toOrigin.SetHop (hop);
1493          toOrigin.SetLifeTime (std::max (Time (2 * m_netTraversalTime - 2 * hop * m_nodeTraversalTime),
1494                                          toOrigin.GetLifeTime ()));
1495          // EB-AODV
1496          toOrigin.SetNextHopXLoc(rreqHeader.GetLocX());
```

```
        toOrigin.SetNextHopYLoc(rreqHeader.GetLocY());
        toOrigin.SetEnergyAccum(rreqHeader.GetEnergyAccum());
        m_routingTable.Update (toOrigin);
        updateRREQTimerValues = true;
      }
    // !!! Why is the rtable entry getting updated from a RREQ with smaller seq.no. than the entry?
    /* For EB-AODV, RREQs are more picky about route than RREPs, should only update route
     * when seq.no. is greater, path energy is greater, or an invalid sequence number
    std::cout<<ipv4Addr<<"Updating routing table with new RREQ... even if seq.no. is smaller..."<<std::endl;
    toOrigin.SetValidSeqNo (true);
    toOrigin.SetNextHop (src);
    toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
    toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
    toOrigin.SetHop (hop);
    toOrigin.SetLifeTime (std::max (Time (2 * m_netTraversalTime - 2 * hop * m_nodeTraversalTime),
                                    toOrigin.GetLifeTime ()));
    // EB-AODV
    toOrigin.SetNextHopXLoc(rreqHeader.GetLocX());
    toOrigin.SetNextHopYLoc(rreqHeader.GetLocY());
    toOrigin.SetEnergyAccum(rreqHeader.GetEnergyAccum());
    m_routingTable.Update (toOrigin);
    //m_nb.Update (src, Time (AllowedHelloLoss * HelloInterval));
    */
  }


RoutingTableEntry toNeighbor;
if (!m_routingTable.LookupRoute (src, toNeighbor))
  {
    NS_LOG_DEBUG ("Neighbor:" << src << "␣not␣found␣in␣routing␣table.␣Creating␣an␣entry");
    Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
    RoutingTableEntry newEntry (
      dev, src, false, rreqHeader.GetOriginSeqno (),
      m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0), 1, src,
      m_activeRouteTimeout, rreqHeader.GetLocX(),rreqHeader.GetLocY(),
      rreqHeader.GetNeighborEnergy()
    );
    m_routingTable.AddRoute (newEntry);
  }
else
  {
    toNeighbor.SetLifeTime (m_activeRouteTimeout);
    toNeighbor.SetValidSeqNo (false);
    toNeighbor.SetSeqNo (rreqHeader.GetOriginSeqno ());
    toNeighbor.SetFlag (VALID);
    toNeighbor.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
    toNeighbor.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
    toNeighbor.SetHop (1);
    toNeighbor.SetNextHop (src);
    // EB-AODV
    toNeighbor.SetNextHopXLoc(rreqHeader.GetLocX());
    toNeighbor.SetNextHopYLoc(rreqHeader.GetLocY());
    toNeighbor.SetEnergyAccum(rreqHeader.GetNeighborEnergy());
    m_routingTable.Update (toNeighbor);
  }
m_nb.Update (src, Time (m_allowedHelloLoss * m_helloInterval));

NS_LOG_LOGIC (receiver << "␣receive␣RREQ␣with␣hop␣count␣" << static_cast<uint32_t> (rreqHeader.GetHopCount
    ↪ ())
                  << "␣ID␣" << rreqHeader.GetId ()
                  << "␣to␣destination␣" << rreqHeader.GetDst ());

// A node generates a RREP if either:
// (i) it is itself the destination,
Ipv4Address dst = rreqHeader.GetDst ();
if (IsMyOwnAddress (rreqHeader.GetDst ()) && updateRREQTimerValues)
  {
    BUGOUT(ipv4Addr<<":␣RREQ␣intended␣for␣me.␣Wait␣for␣more..,");
```

```
    m_routingTable.LookupRoute (origin, toOrigin);
    // EB-AODV start timer to wait for multiple RREQs
    //m_addressReqTimer[dst].Remove ();
    //m_addressReqTimer.erase (dst);
    //m_rreqRateLimitTimer.SetFunction (&RoutingProtocol::RreqRateLimitTimerExpire,
    // this);
    if (m_addressReqTimer.find (dst) == m_addressReqTimer.end ()
      && !(m_multipleRreqTimer[dst].IsRunning())){ // if timer doesn't exist
      BUGOUT("Timer doesn't exist. Creating one...");
      if (m_multipleRreqTimer[dst].IsSuspended() || m_multipleRreqTimer[dst].IsExpired())
        m_multipleRreqTimer[dst].Remove();
      m_multipleRreqTimer[dst].SetFunction(&RoutingProtocol::MultipleRREQRecvTimerExpire,this);
      m_multipleRreqTimer[dst].SetArguments(dst,rreqHeader,toOrigin);
      m_multipleRreqTimer[dst].Schedule(Time (MilliSeconds (RREQ_WAIT_TIME)));
    }
    // update values being sent to RREQ timer function if energy is better
    Time timeLeft = m_multipleRreqTimer[dst].GetDelayLeft();
    m_multipleRreqTimer[dst].Remove ();
    m_multipleRreqTimer[dst].SetFunction(&RoutingProtocol::MultipleRREQRecvTimerExpire,this);
    m_multipleRreqTimer[dst].SetArguments(dst,rreqHeader,toOrigin);
    m_multipleRreqTimer[dst].Schedule(timeLeft);
    //SendReply (rreqHeader, toOrigin);
    BUGOUT(ipv4Addr<<": saving route with next hop: "<<toOrigin.GetNextHop());
    return;
  }
/*
 * (ii) or it has an active route to the destination, the destination sequence number in the node's
     ↪ existing route table entry for the destination
 * is valid and greater than or equal to the Destination Sequence Number of the RREQ, and the "destination
     ↪ only" flag is NOT set.
 */
RoutingTableEntry toDst;
//Ipv4Address dst = rreqHeader.GetDst ();
if (m_routingTable.LookupRoute (dst, toDst))
  {
    /*
     * Drop RREQ, This node RREP will make a loop.
     */
    if (toDst.GetNextHop () == src)
      {
        BUGOUT(ipv4Addr<<": Drop RREQ, will make loop");
        NS_LOG_DEBUG ("Drop RREQ from " << src << ", dest next hop " << toDst.GetNextHop ());
        return;
      }
    /*
     * The Destination Sequence number for the requested destination is set to the maximum of the
         ↪ corresponding value
     * received in the RREQ message, and the destination sequence value currently maintained by the node for
         ↪ the requested destination.
     * However, the forwarding node MUST NOT modify its maintained value for the destination sequence
         ↪ number, even if the value
     * received in the incoming RREQ is larger than the value currently maintained by the forwarding node.
     */
    if ((rreqHeader.GetUnknownSeqno () || (int32_t (toDst.GetSeqNo ()) - int32_t (rreqHeader.GetDstSeqno ())
        ↪ >= 0))
       && toDst.GetValidSeqNo () )
      {
        if (!rreqHeader.GetDestinationOnly () && toDst.GetFlag () == VALID)
          {
            m_routingTable.LookupRoute (origin, toOrigin);
            SendReplyByIntermediateNode (toDst, toOrigin, rreqHeader.GetGratuitousRrep ());
            return;
          }
        rreqHeader.SetDstSeqno (toDst.GetSeqNo ());
        rreqHeader.SetUnknownSeqno (false);
      }
  }

```

```
  SocketIpTtlTag tag;
  p->RemovePacketTag (tag);
  if (tag.GetTtl () < 2)
    {
      NS_LOG_DEBUG ("TTL␣exceeded.␣Drop␣RREQ␣origin␣" << src << "␣destination␣" << dst );
      return;
    }

  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
        m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
    {
      Ptr<Socket> socket = j->first;
      Ipv4InterfaceAddress iface = j->second;
      Ptr<Packet> packet = Create<Packet> ();
      SocketIpTtlTag ttl;
      ttl.SetTtl (tag.GetTtl () - 1);
    rreqHeader.SetEnergyAccum(this->m_energyLevel + rreqHeader.GetEnergyAccum()); // Add current node's power to header accumulated power.
    rreqHeader.SetNeighborEnergy(this->m_energyLevel);
      getCoordinates(); // Refresh coordinates
      rreqHeader.SetLocX(this->nodeX); // Commit coordinates to header
      rreqHeader.SetLocY(this->nodeY);
      packet->AddPacketTag (ttl);
      packet->AddHeader (rreqHeader);
      TypeHeader tHeader (LS_AODVTYPE_RREQ);
      packet->AddHeader (tHeader);
      // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
      Ipv4Address destination;
      if (iface.GetMask () == Ipv4Mask::GetOnes ())
        {
          destination = Ipv4Address ("255.255.255.255");
        }
      else
        {
          destination = iface.GetBroadcast ();
        }
      m_lastBcastTime = Simulator::Now ();
      Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))), &RoutingProtocol::SendTo, this, socket, packet, destination);

    }
}

void
RoutingProtocol::SendReply (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin)
{
  NS_LOG_FUNCTION (this << toOrigin.GetDestination ());
  /*
   * Destination node MUST increment its own sequence number by one if the sequence number in the RREQ packet is equal to that
   * incremented value. Otherwise, the destination does not change its sequence number before generating the RREP message.
   */
  if (!rreqHeader.GetUnknownSeqno () && (rreqHeader.GetDstSeqno () == m_seqNo + 1))
    {
      m_seqNo++;
    }
  // EA-AODV get node position
  this->getCoordinates();
  RrepHeader rrepHeader ( /*prefixSize=*/ 0, /*hops=*/ 0, /*dst=*/ rreqHeader.GetDst (),
                          /*dstSeqNo=*/ m_seqNo, /*origin=*/ toOrigin.GetDestination (),
                          /*lifeTime=*/ m_myRouteTimeout, nodeX, nodeY);
  Ptr<Packet> packet = Create<Packet> ();
  SocketIpTtlTag tag;
  tag.SetTtl (toOrigin.GetHop ());
  packet->AddPacketTag (tag);
  packet->AddHeader (rrepHeader);
  TypeHeader tHeader (LS_AODVTYPE_RREP);
```

```
  packet->AddHeader (tHeader);
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
  NS_ASSERT (socket);
  // Energy Model
  BUGOUT(ipv4Addr<<": power decay for transmission during route discovery at time:"<<Simulator::Now()<<" to "<<toOrigin.GetNextHop());
  m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toOrigin),PATH_LOSS_EXP);
  socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), LS_AODV_PORT));
}

void
RoutingProtocol::SendReplyByIntermediateNode (RoutingTableEntry & toDst, RoutingTableEntry & toOrigin, bool gratRep)
{
  NS_LOG_FUNCTION (this);
  RrepHeader rrepHeader (/*prefix size=*/ 0, /*hops=*/ toDst.GetHop (),
         /*dst=*/ toDst.GetDestination (), /*dst seqno=*/ toDst.GetSeqNo (),
         /*origin=*/ toOrigin.GetDestination (), /*lifetime=*/ toDst.GetLifeTime (),
         toOrigin.GetNextHopXLoc(), toOrigin.GetNextHopYLoc());
         // not sure if X and Y locations should be from origin or destination???
  /* If the node we received a RREQ for is a neighbor we are
   * probably facing a unidirectional link... Better request a RREP-ack
   */
  if (toDst.GetHop () == 1)
    {
      rrepHeader.SetAckRequired (true);
      RoutingTableEntry toNextHop;
      m_routingTable.LookupRoute (toOrigin.GetNextHop (), toNextHop);
      toNextHop.m_ackTimer.SetFunction (&RoutingProtocol::AckTimerExpire, this);
      toNextHop.m_ackTimer.SetArguments (toNextHop.GetDestination (), m_blackListTimeout);
      toNextHop.m_ackTimer.SetDelay (m_nextHopWait);
    }
  toDst.InsertPrecursor (toOrigin.GetNextHop ());
  toOrigin.InsertPrecursor (toDst.GetNextHop ());
  m_routingTable.Update (toDst);
  m_routingTable.Update (toOrigin);

  Ptr<Packet> packet = Create<Packet> ();
  SocketIpTtlTag tag;
  tag.SetTtl (toOrigin.GetHop ());
  packet->AddPacketTag (tag);
  packet->AddHeader (rrepHeader);
  TypeHeader tHeader (LS_AODVTYPE_RREP);
  packet->AddHeader (tHeader);
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
  NS_ASSERT (socket);
  // Energy Model
  BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
  m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toOrigin),PATH_LOSS_EXP);
  socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), LS_AODV_PORT));

  // Generating gratuitous RREPs
  if (gratRep)
    {
      RrepHeader gratRepHeader (/*prefix size=*/ 0, /*hops=*/ toOrigin.GetHop (), /*dst=*/ toOrigin.GetDestination (),
                                          /*dst seqno=*/ toOrigin.GetSeqNo (), /*origin=*/ toDst.GetDestination (),
                                          /*lifetime=*/ toOrigin.GetLifeTime ());
      Ptr<Packet> packetToDst = Create<Packet> ();
      SocketIpTtlTag gratTag;
      gratTag.SetTtl (toDst.GetHop ());
      packetToDst->AddPacketTag (gratTag);
      packetToDst->AddHeader (gratRepHeader);
      TypeHeader type (LS_AODVTYPE_RREP);
      packetToDst->AddHeader (type);
      Ptr<Socket> socket = FindSocketWithInterfaceAddress (toDst.GetInterface ());
      NS_ASSERT (socket);
```

```
      NS_LOG_LOGIC ("Send␣gratuitous␣RREP␣" << packet->GetUid ());
      // Energy Model
      BUGOUT(ipv4Addr<<":␣power␣decay␣for␣transmission␣during␣route␣discovery.");
      m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toDst),PATH_LOSS_EXP);
      socket->SendTo (packetToDst, 0, InetSocketAddress (toDst.GetNextHop (), LS_AODV_PORT));
    }
}

void
RoutingProtocol::SendReplyAck (Ipv4Address neighbor)
{
  NS_LOG_FUNCTION (this << "␣to␣" << neighbor);
  RrepAckHeader h;
  TypeHeader typeHeader (LS_AODVTYPE_RREP_ACK);
  Ptr<Packet> packet = Create<Packet> ();
  SocketIpTtlTag tag;
  tag.SetTtl (1);
  packet->AddPacketTag (tag);
  packet->AddHeader (h);
  packet->AddHeader (typeHeader);
  RoutingTableEntry toNeighbor;
  m_routingTable.LookupRoute (neighbor, toNeighbor);
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (toNeighbor.GetInterface ());
  NS_ASSERT (socket);
  // Energy Model
  BUGOUT(ipv4Addr<<":␣power␣decay␣for␣transmission␣during␣route␣discovery.");
  m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toNeighbor),PATH_LOSS_EXP);
  socket->SendTo (packet, 0, InetSocketAddress (neighbor, LS_AODV_PORT));
}

void
RoutingProtocol::RecvReply (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address sender)
{
  NS_LOG_FUNCTION (this << "␣src␣" << sender);
  RrepHeader rrepHeader;
  p->RemoveHeader (rrepHeader);
  Ipv4Address dst = rrepHeader.GetDst ();
  NS_LOG_LOGIC ("RREP␣destination␣" << dst << "␣RREP␣origin␣" << rrepHeader.GetOrigin ());

  uint8_t hop = rrepHeader.GetHopCount () + 1;
  rrepHeader.SetHopCount (hop);

  // If RREP is Hello message
  if (dst == rrepHeader.GetOrigin ())
    {
      ProcessHello (rrepHeader, receiver);
      return;
    }

  /*
   * If the route table entry to the destination is created or updated, then the following actions occur:
   * - the route is marked as active,
   * - the destination sequence number is marked as valid,
   * - the next hop in the route entry is assigned to be the node from which the RREP is received,
   * which is indicated by the source IP address field in the IP header,
   * - the hop count is set to the value of the hop count from RREP message + 1
   * - the expiry time is set to the current time plus the value of the Lifetime in the RREP message,
   * - and the destination sequence number is the Destination Sequence Number in the RREP message.
   */
  Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
  RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ dst, /*validSeqNo=*/ true, /*seqno=*/
      ↪ rrepHeader.GetDstSeqno (),
                                  /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                      ↪ (receiver), 0),/*hop=*/ hop,
                                  /*nextHop=*/ sender, /*lifeTime=*/ rrepHeader.GetLifeTime (),
                                  rrepHeader.GetLocX(), rrepHeader.GetLocY());
  RoutingTableEntry toDst;
  // EA-AODV get distance to node that received RREP from
```

```
1820  //float rrepDistance = sqrt(pow(rrepHeader.GetLocX(),2) + pow(rrepHeader.GetLocY(),2));
1821  if (m_routingTable.LookupRoute (dst, toDst))
1822    {
1823      // get distance to next hop neighbor on current
1824      //float entryDistance = sqrt(pow(toDst.GetNextHopXLoc(),2) + pow(toDst.GetNextHopYLoc(),2));
1825      /*
1826       * The existing entry is updated only in the following circumstances:
1827       * (i) the sequence number in the routing table is marked as invalid in route table entry.
1828       */
1829      if (!toDst.GetValidSeqNo ())
1830        {
1831          m_routingTable.Update (newEntry);
1832        }
1833      // (ii)the Destination Sequence Number in the RREP is greater than the node's copy of the destination
           ↪ sequence number and the known value is valid,
1834      else if ((int32_t (rrepHeader.GetDstSeqno ()) - int32_t (toDst.GetSeqNo ())) > 0)
1835        {
1836          m_routingTable.Update (newEntry);
1837        }
1838      else if ((rrepHeader.GetDstSeqno () == toDst.GetSeqNo ()))
1839        {
1840          // (iii) the sequence numbers are the same, but the route is marked as inactive.
1841          if (/*(rrepHeader.GetDstSeqno () == toDst.GetSeqNo ()) &&*/ (toDst.GetFlag () != VALID))
1842            {
1843              m_routingTable.Update (newEntry);
1844            }
1845          /*
1846          // EA-AODV if distance to neighbor that sent RREP is smaller, choose that path
1847          else if (rrepDistance < entryDistance) {
1848            m_routingTable.Update(newEntry);
1849          }*/
1850          // if distance for current path is smaller and...
1851          // (iv) the sequence numbers are the same, and the New Hop Count is smaller than the hop count in
               ↪ route table entry.
1852          else if ((rrepHeader.GetDstSeqno () == toDst.GetSeqNo ()) && (hop < toDst.GetHop ()))
1853            {
1854              m_routingTable.Update (newEntry);
1855            }
1856        }
1857    }
1858  else
1859    {
1860      // The forward route for this destination is created if it does not already exist.
1861      NS_LOG_LOGIC ("add␣new␣route");
1862      m_routingTable.AddRoute (newEntry);
1863    }
1864  // Acknowledge receipt of the RREP by sending a RREP-ACK message back
1865  if (rrepHeader.GetAckRequired ())
1866    {
1867      SendReplyAck (sender);
1868      rrepHeader.SetAckRequired (false);
1869    }
1870  NS_LOG_LOGIC ("receiver␣" << receiver << "␣origin␣" << rrepHeader.GetOrigin ());
1871  if (IsMyOwnAddress (rrepHeader.GetOrigin ()))
1872    {
1873      if (toDst.GetFlag () == IN_SEARCH)
1874        {
1875          m_routingTable.Update (newEntry);
1876          m_addressReqTimer[dst].Remove ();
1877          m_addressReqTimer.erase (dst);
1878        }
1879      m_routingTable.LookupRoute (dst, toDst);
1880      // EB-AODV
1881      // Was waiting for RREP from this destination, now send data to destination
1882      SendPacketFromQueue (dst, toDst.GetRoute ());
1883      return;
1884    }
1885
```

```
  RoutingTableEntry toOrigin;
  if (!m_routingTable.LookupRoute (rrepHeader.GetOrigin (), toOrigin) || toOrigin.GetFlag () == IN_SEARCH)
    {
      return; // Impossible! drop.
    }
  toOrigin.SetLifeTime (std::max (m_activeRouteTimeout, toOrigin.GetLifeTime ()));
  m_routingTable.Update (toOrigin);

  // Update information about precursors
  if (m_routingTable.LookupValidRoute (rrepHeader.GetDst (), toDst))
    {
      toDst.InsertPrecursor (toOrigin.GetNextHop ());
      m_routingTable.Update (toDst);

      RoutingTableEntry toNextHopToDst;
      m_routingTable.LookupRoute (toDst.GetNextHop (), toNextHopToDst);
      toNextHopToDst.InsertPrecursor (toOrigin.GetNextHop ());
      m_routingTable.Update (toNextHopToDst);

      toOrigin.InsertPrecursor (toDst.GetNextHop ());
      m_routingTable.Update (toOrigin);

      RoutingTableEntry toNextHopToOrigin;
      m_routingTable.LookupRoute (toOrigin.GetNextHop (), toNextHopToOrigin);
      toNextHopToOrigin.InsertPrecursor (toDst.GetNextHop ());
      m_routingTable.Update (toNextHopToOrigin);
    }
  SocketIpTtlTag tag;
  p->RemovePacketTag (tag);
  if (tag.GetTtl () < 2)
    {
      NS_LOG_DEBUG ("TTL␣exceeded.␣Drop␣RREP␣destination␣" << dst << "␣origin␣" << rrepHeader.GetOrigin ());
      return;
    }

  Ptr<Packet> packet = Create<Packet> ();
  SocketIpTtlTag ttl;
  ttl.SetTtl (tag.GetTtl () - 1);
  packet->AddPacketTag (ttl);
  packet->AddHeader (rrepHeader);
  TypeHeader tHeader (LS_AODVTYPE_RREP);
  packet->AddHeader (tHeader);
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
  NS_ASSERT (socket);
  // Energy Model
  BUGOUT(ipv4Addr<<":␣power␣decay␣for␣transmission␣during␣route␣discovery.");
  m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toOrigin),PATH_LOSS_EXP);
  socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), LS_AODV_PORT));
}

void
RoutingProtocol::RecvReplyAck (Ipv4Address neighbor)
{
  NS_LOG_FUNCTION (this);
  RoutingTableEntry rt;
  if (m_routingTable.LookupRoute (neighbor, rt))
    {
      rt.m_ackTimer.Cancel ();
      rt.SetFlag (VALID);
      m_routingTable.Update (rt);
    }
}

void
RoutingProtocol::ProcessHello (RrepHeader const & rrepHeader, Ipv4Address receiver )
{
  NS_LOG_FUNCTION (this << "from␣" << rrepHeader.GetDst ());
  /*
```

```
   * Whenever a node receives a Hello message from a neighbor, the node
   * SHOULD make sure that it has an active route to the neighbor, and
   * create one if necessary.
   */
  RoutingTableEntry toNeighbor;
  if (!m_routingTable.LookupRoute (rrepHeader.GetDst (), toNeighbor))
    {
      Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
      RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ rrepHeader.GetDst (), /*validSeqNo=*/ true,
          ↪ /*seqno=*/ rrepHeader.GetDstSeqno (),
                                  /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress
                                      ↪ (receiver), 0),
                                  /*hop=*/ 1, /*nextHop=*/ rrepHeader.GetDst (), /*lifeTime=*/
                                      ↪ rrepHeader.GetLifeTime ());
      m_routingTable.AddRoute (newEntry);
    }
  else
    {
      toNeighbor.SetLifeTime (std::max (Time (m_allowedHelloLoss * m_helloInterval), toNeighbor.GetLifeTime
          ↪ ()));
      toNeighbor.SetSeqNo (rrepHeader.GetDstSeqno ());
      toNeighbor.SetValidSeqNo (true);
      toNeighbor.SetFlag (VALID);
      toNeighbor.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
      toNeighbor.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
      toNeighbor.SetHop (1);
      toNeighbor.SetNextHop (rrepHeader.GetDst ());
      m_routingTable.Update (toNeighbor);
    }
  if (m_enableHello)
    {
      m_nb.Update (rrepHeader.GetDst (), Time (m_allowedHelloLoss * m_helloInterval));
    }
}

void
RoutingProtocol::RecvError (Ptr<Packet> p, Ipv4Address src )
{
  NS_LOG_FUNCTION (this << "␣from␣" << src);
  RerrHeader rerrHeader;
  p->RemoveHeader (rerrHeader);
  std::map<Ipv4Address, uint32_t> dstWithNextHopSrc;
  std::map<Ipv4Address, uint32_t> unreachable;
  m_routingTable.GetListOfDestinationWithNextHop (src, dstWithNextHopSrc);
  std::pair<Ipv4Address, uint32_t> un;
  while (rerrHeader.RemoveUnDestination (un))
    {
      for (std::map<Ipv4Address, uint32_t>::const_iterator i =
             dstWithNextHopSrc.begin (); i != dstWithNextHopSrc.end (); ++i)
        {
          if (i->first == un.first)
            {
              unreachable.insert (un);
            }
        }
    }

  std::vector<Ipv4Address> precursors;
  for (std::map<Ipv4Address, uint32_t>::const_iterator i = unreachable.begin ();
       i != unreachable.end (); )
    {
      if (!rerrHeader.AddUnDestination (i->first, i->second))
        {
          TypeHeader typeHeader (LS_AODVTYPE_RERR);
          Ptr<Packet> packet = Create<Packet> ();
          SocketIpTtlTag tag;
          tag.SetTtl (1);
          packet->AddPacketTag (tag);
```

```
2018          packet->AddHeader (rerrHeader);
2019          packet->AddHeader (typeHeader);
2020          SendRerrMessage (packet, precursors);
2021          rerrHeader.Clear ();
2022        }
2023      else
2024        {
2025          RoutingTableEntry toDst;
2026          m_routingTable.LookupRoute (i->first, toDst);
2027          toDst.GetPrecursors (precursors);
2028          ++i;
2029        }
2030    }
2031  if (rerrHeader.GetDestCount () != 0)
2032    {
2033      TypeHeader typeHeader (LS_AODVTYPE_RERR);
2034      Ptr<Packet> packet = Create<Packet> ();
2035      SocketIpTtlTag tag;
2036      tag.SetTtl (1);
2037      packet->AddPacketTag (tag);
2038      packet->AddHeader (rerrHeader);
2039      packet->AddHeader (typeHeader);
2040      SendRerrMessage (packet, precursors);
2041    }
2042  m_routingTable.InvalidateRoutesWithDst (unreachable);
2043 }
2044
2045 void
2046 RoutingProtocol::RouteRequestTimerExpire (Ipv4Address dst)
2047 {
2048  NS_LOG_LOGIC (this);
2049  RoutingTableEntry toDst;
2050  if (m_routingTable.LookupValidRoute (dst, toDst))
2051    {
2052      SendPacketFromQueue (dst, toDst.GetRoute ());
2053      NS_LOG_LOGIC ("route␣to␣" << dst << "␣found");
2054      return;
2055    }
2056  /*
2057   * If a route discovery has been attempted RreqRetries times at the maximum TTL without
2058   * receiving any RREP, all data packets destined for the corresponding destination SHOULD be
2059   * dropped from the buffer and a Destination Unreachable message SHOULD be delivered to the application.
2060   */
2061  if (toDst.GetRreqCnt () == m_rreqRetries)
2062    {
2063      NS_LOG_LOGIC ("route␣discovery␣to␣" << dst << "␣has␣been␣attempted␣RreqRetries␣(" << m_rreqRetries << ")␣
           ↪ times␣with␣ttl␣" << m_netDiameter);
2064      m_addressReqTimer.erase (dst);
2065      m_routingTable.DeleteRoute (dst);
2066      NS_LOG_DEBUG ("Route␣not␣found.␣Drop␣all␣packets␣with␣dst␣" << dst);
2067      m_queue.DropPacketWithDst (dst);
2068      if (isReachable == true) {
2069        SIMOUT("Node␣"<<ipv4Addr<<"␣unreachable␣at␣time␣"<<Simulator::Now().GetMilliSeconds()<<"ms.");
2070        isReachable = false;
2071      }
2072      return;
2073    }
2074
2075  if (toDst.GetFlag () == IN_SEARCH)
2076    {
2077      NS_LOG_LOGIC ("Resend␣RREQ␣to␣" << dst << "␣previous␣ttl␣" << toDst.GetHop ());
2078      SendRequest (dst);
2079    }
2080  else
2081    {
2082      NS_LOG_DEBUG ("Route␣down.␣Stop␣search.␣Drop␣packet␣with␣destination␣" << dst);
2083      m_addressReqTimer.erase (dst);
2084      m_routingTable.DeleteRoute (dst);
```

```
      m_queue.DropPacketWithDst (dst);
      if (isReachable == true) {
        SIMOUT("Node␣"<<ipv4Addr<<"␣unreachable␣at␣time␣"<<Simulator::Now().GetMilliSeconds()<<"ms.");
        isReachable = false;
      }
    }
}

void
RoutingProtocol::HelloTimerExpire ()
{
  NS_LOG_FUNCTION (this);
  Time offset = Time (Seconds (0));
  if (m_lastBcastTime > Time (Seconds (0)))
    {
      offset = Simulator::Now () - m_lastBcastTime;
      NS_LOG_DEBUG ("Hello␣deferred␣due␣to␣last␣bcast␣at:" << m_lastBcastTime);
    }
  else
    {
      SendHello ();
    }
  m_htimer.Cancel ();
  Time diff = m_helloInterval - offset;
  m_htimer.Schedule (std::max (Time (Seconds (0)), diff));
  m_lastBcastTime = Time (Seconds (0));
}

void
RoutingProtocol::RreqRateLimitTimerExpire ()
{
  NS_LOG_FUNCTION (this);
  m_rreqCount = 0;
  m_rreqRateLimitTimer.Schedule (Seconds (1));
}

void
RoutingProtocol::RerrRateLimitTimerExpire ()
{
  NS_LOG_FUNCTION (this);
  m_rerrCount = 0;
  m_rerrRateLimitTimer.Schedule (Seconds (1));
}

// LS_AODV used to decay energy level periodically based on idle time
// and antenna receive power
void RoutingProtocol::PowerDecayTimerExpire() {
  m_energyLevel -= (power_active+power_receive)*0.002;
  if (m_energyLevel < m_aliveThreshold) {
    Ipv4Address ipv4Addr = m_ipv4->GetAddress (1, 0).GetLocal ();
    getCoordinates();
    SIMOUT("Node␣"<<ipv4Addr<<"␣␣died␣at␣time␣"<<Simulator::Now().GetMilliSeconds()<<"ms.");
    m_powerDecayTimer.Cancel();
  } else {
    m_powerDecayTimer.Schedule(MilliSeconds(2));
  }
}

// LS_AODV used to transmit a reply to RREQ after collecting for a period of time
void RoutingProtocol::MultipleRREQRecvTimerExpire(Ipv4Address dst, RreqHeader rreqHeader,
    RoutingTableEntry toOrigin) {

  m_multipleRreqTimer.erase (dst);
  // need some way to pass rreqHeader and toOrigin variables from RecvRequest()
  BUGOUT(ipv4Addr<<":␣RREQ␣timeout␣elapsed.␣Sending␣reply␣to␣"<<dst<<"␣with␣next␣
      ↪ hop:"<<toOrigin.GetNextHop());
  SendReply (rreqHeader, toOrigin);
}
```

```
2152
2153 void
2154 RoutingProtocol::AckTimerExpire (Ipv4Address neighbor, Time blacklistTimeout)
2155 {
2156   NS_LOG_FUNCTION (this);
2157   m_routingTable.MarkLinkAsUnidirectional (neighbor, blacklistTimeout);
2158 }
2159
2160 void
2161 RoutingProtocol::SendHello ()
2162 {
2163   NS_LOG_FUNCTION (this);
2164   /* Broadcast a RREP with TTL = 1 with the RREP message fields set as follows:
2165    * Destination IP Address The node's IP address.
2166    * Destination Sequence Number The node's latest sequence number.
2167    * Hop Count 0
2168    * Lifetime AllowedHelloLoss * HelloInterval
2169    */
2170   for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j = m_socketAddresses.begin (); j !=
         ↪ m_socketAddresses.end (); ++j)
2171     {
2172       Ptr<Socket> socket = j->first;
2173       Ipv4InterfaceAddress iface = j->second;
2174       RrepHeader helloHeader (/*prefix size=*/ 0, /*hops=*/ 0, /*dst=*/ iface.GetLocal (), /*dst seqno=*/
             ↪ m_seqNo,
2175                               /*origin=*/ iface.GetLocal (),/*lifetime=*/ Time
                                     ↪ (m_allowedHelloLoss * m_helloInterval));
2176       Ptr<Packet> packet = Create<Packet> ();
2177       SocketIpTtlTag tag;
2178       tag.SetTtl (1);
2179       packet->AddPacketTag (tag);
2180       packet->AddHeader (helloHeader);
2181       TypeHeader tHeader (LS_AODVTYPE_RREP);
2182       packet->AddHeader (tHeader);
2183       // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
2184       Ipv4Address destination;
2185       BUGOUT(ipv4Addr<<": sending Hello message to: "<<destination);
2186       if (iface.GetMask () == Ipv4Mask::GetOnes ())
2187         {
2188           destination = Ipv4Address ("255.255.255.255");
2189         }
2190       else
2191         {
2192           destination = iface.GetBroadcast ();
2193         }
2194       Time jitter = Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10)));
2195       Simulator::Schedule (jitter, &RoutingProtocol::SendTo, this, socket, packet, destination);
2196     }
2197 }
2198
2199 void
2200 RoutingProtocol::SendPacketFromQueue (Ipv4Address dst, Ptr<Ipv4Route> route)
2201 {
2202   NS_LOG_FUNCTION (this);
2203   QueueEntry queueEntry;
2204   while (m_queue.Dequeue (dst, queueEntry))
2205     {
2206       DeferredRouteOutputTag tag;
2207       Ptr<Packet> p = ConstCast<Packet> (queueEntry.GetPacket ());
2208       if (p->RemovePacketTag (tag)
2209           && tag.GetInterface () != -1
2210           && tag.GetInterface () != m_ipv4->GetInterfaceForDevice (route->GetOutputDevice ()))
2211         {
2212           NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
2213           return;
2214         }
2215       UnicastForwardCallback ucb = queueEntry.GetUnicastForwardCallback ();
2216       Ipv4Header header = queueEntry.GetIpv4Header ();
```

```
2217      header.SetSource (route->GetSource ());
2218      header.SetTtl (header.GetTtl () + 1); // compensate extra TTL decrement by fake loopback routing
2219      ucb (route, p, header);
2220
2221      // Energy section
2222      this->getCoordinates();
2223      m_lastUpdateTime = Simulator::Now();
2224      RoutingTableEntry rt;
2225      m_routingTable.LookupRoute(dst,rt);
2226      m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(rt),PATH_LOSS_EXP);
2227    }
2228  }
2229
2230  void
2231  RoutingProtocol::SendRerrWhenBreaksLinkToNextHop (Ipv4Address nextHop)
2232  {
2233    NS_LOG_FUNCTION (this << nextHop);
2234    if (m_energyLevel > m_aliveThreshold) {
2235      RerrHeader rerrHeader;
2236      std::vector<Ipv4Address> precursors;
2237      std::map<Ipv4Address, uint32_t> unreachable;
2238
2239      RoutingTableEntry toNextHop;
2240      if (!m_routingTable.LookupRoute (nextHop, toNextHop))
2241        {
2242          return;
2243        }
2244      toNextHop.GetPrecursors (precursors);
2245      rerrHeader.AddUnDestination (nextHop, toNextHop.GetSeqNo ());
2246      m_routingTable.GetListOfDestinationWithNextHop (nextHop, unreachable);
2247      for (std::map<Ipv4Address, uint32_t>::const_iterator i = unreachable.begin (); i
2248           != unreachable.end (); )
2249        {
2250          if (!rerrHeader.AddUnDestination (i->first, i->second))
2251            {
2252              NS_LOG_LOGIC ("Send␣RERR␣message␣with␣maximum␣size.");
2253              TypeHeader typeHeader (LS_AODVTYPE_RERR);
2254              Ptr<Packet> packet = Create<Packet> ();
2255              SocketIpTtlTag tag;
2256              tag.SetTtl (1);
2257              packet->AddPacketTag (tag);
2258              packet->AddHeader (rerrHeader);
2259              packet->AddHeader (typeHeader);
2260              SendRerrMessage (packet, precursors);
2261              rerrHeader.Clear ();
2262            }
2263          else
2264            {
2265              RoutingTableEntry toDst;
2266              m_routingTable.LookupRoute (i->first, toDst);
2267              toDst.GetPrecursors (precursors);
2268              ++i;
2269            }
2270        }
2271      if (rerrHeader.GetDestCount () != 0)
2272        {
2273          TypeHeader typeHeader (LS_AODVTYPE_RERR);
2274          Ptr<Packet> packet = Create<Packet> ();
2275          SocketIpTtlTag tag;
2276          tag.SetTtl (1);
2277          packet->AddPacketTag (tag);
2278          packet->AddHeader (rerrHeader);
2279          packet->AddHeader (typeHeader);
2280          SendRerrMessage (packet, precursors);
2281        }
2282      unreachable.insert (std::make_pair (nextHop, toNextHop.GetSeqNo ()));
2283      m_routingTable.InvalidateRoutesWithDst (unreachable);
2284    }
```

```
}

void
RoutingProtocol::SendRerrWhenNoRouteToForward (Ipv4Address dst,
                                               uint32_t dstSeqNo, Ipv4Address origin)
{
  NS_LOG_FUNCTION (this);
  if (m_energyLevel > m_aliveThreshold) {
    // A node SHOULD NOT originate more than RERR_RATELIMIT RERR messages per second.
    if (m_rerrCount == m_rerrRateLimit)
      {
        // Just make sure that the RerrRateLimit timer is running and will expire
        NS_ASSERT (m_rerrRateLimitTimer.IsRunning ());
        // discard the packet and return
        NS_LOG_LOGIC ("RerrRateLimit reached at " << Simulator::Now ().GetSeconds () << " with timer delay
            ↪ left "
                                                  << m_rerrRateLimitTimer.GetDelayLeft ().GetSeconds ()
                                                  << "; suppressing RERR");
        return;
      }
    RerrHeader rerrHeader;
    rerrHeader.AddUnDestination (dst, dstSeqNo);
    RoutingTableEntry toOrigin;
    Ptr<Packet> packet = Create<Packet> ();
    SocketIpTtlTag tag;
    tag.SetTtl (1);
    packet->AddPacketTag (tag);
    packet->AddHeader (rerrHeader);
    packet->AddHeader (TypeHeader (LS_AODVTYPE_RERR));
    if (m_routingTable.LookupValidRoute (origin, toOrigin))
      {
        Ptr<Socket> socket = FindSocketWithInterfaceAddress (
            toOrigin.GetInterface ());
        NS_ASSERT (socket);
        NS_LOG_LOGIC ("Unicast RERR to the source of the data transmission");
        // Energy Model
        BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
        m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(toOrigin),PATH_LOSS_EXP);
        socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), LS_AODV_PORT));
      }
    else
      {
        for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator i =
               m_socketAddresses.begin (); i != m_socketAddresses.end (); ++i)
          {
            Ptr<Socket> socket = i->first;
            Ipv4InterfaceAddress iface = i->second;
            NS_ASSERT (socket);
            NS_LOG_LOGIC ("Broadcast RERR message from interface " << iface.GetLocal ());
            // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
            Ipv4Address destination;
            if (iface.GetMask () == Ipv4Mask::GetOnes ())
              {
                destination = Ipv4Address ("255.255.255.255");
              }
            else
              {
                destination = iface.GetBroadcast ();
              }
            // Energy Model
            RoutingTableEntry rt;
            m_routingTable.LookupRoute(destination,rt);
            BUGOUT(ipv4Addr<<": power decay for transmission during route discovery.");
            m_energyLevel -= TxConst * powf(GetDistanceBetweenNodes(rt),PATH_LOSS_EXP);
            socket->SendTo (packet->Copy (), 0, InetSocketAddress (destination, LS_AODV_PORT));
          }
      }
  }
```

```
2352 }
2353
2354 void
2355 RoutingProtocol::SendRerrMessage (Ptr<Packet> packet, std::vector<Ipv4Address> precursors)
2356 {
2357   NS_LOG_FUNCTION (this);
2358   if (m_energyLevel > m_aliveThreshold) {
2359     if (precursors.empty ())
2360       {
2361         NS_LOG_LOGIC ("No precursors");
2362         return;
2363       }
2364     // A node SHOULD NOT originate more than RERR_RATELIMIT RERR messages per second.
2365     if (m_rerrCount == m_rerrRateLimit)
2366       {
2367         // Just make sure that the RerrRateLimit timer is running and will expire
2368         NS_ASSERT (m_rerrRateLimitTimer.IsRunning ());
2369         // discard the packet and return
2370         NS_LOG_LOGIC ("RerrRateLimit reached at " << Simulator::Now ().GetSeconds () << " with timer delay
              ↪ left "
2371                                                   << m_rerrRateLimitTimer.GetDelayLeft ().GetSeconds ()
2372                                                   << "; suppressing RERR");
2373         return;
2374       }
2375     // If there is only one precursor, RERR SHOULD be unicast toward that precursor
2376     if (precursors.size () == 1)
2377       {
2378         RoutingTableEntry toPrecursor;
2379         if (m_routingTable.LookupValidRoute (precursors.front (), toPrecursor))
2380           {
2381             Ptr<Socket> socket = FindSocketWithInterfaceAddress (toPrecursor.GetInterface ());
2382             NS_ASSERT (socket);
2383             NS_LOG_LOGIC ("one precursor => unicast RERR to " << toPrecursor.GetDestination () << " from " <<
                  ↪ toPrecursor.GetInterface ().GetLocal ());
2384             Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
                  ↪ &RoutingProtocol::SendTo, this, socket, packet, precursors.front ());
2385             m_rerrCount++;
2386           }
2387         return;
2388       }
2389
2390     // Should only transmit RERR on those interfaces which have precursor nodes for the broken route
2391     std::vector<Ipv4InterfaceAddress> ifaces;
2392     RoutingTableEntry toPrecursor;
2393     for (std::vector<Ipv4Address>::const_iterator i = precursors.begin (); i != precursors.end (); ++i)
2394       {
2395         if (m_routingTable.LookupValidRoute (*i, toPrecursor)
2396             && std::find (ifaces.begin (), ifaces.end (), toPrecursor.GetInterface ()) == ifaces.end ())
2397           {
2398             ifaces.push_back (toPrecursor.GetInterface ());
2399           }
2400       }
2401
2402     for (std::vector<Ipv4InterfaceAddress>::const_iterator i = ifaces.begin (); i != ifaces.end (); ++i)
2403       {
2404         Ptr<Socket> socket = FindSocketWithInterfaceAddress (*i);
2405         NS_ASSERT (socket);
2406         NS_LOG_LOGIC ("Broadcast RERR message from interface " << i->GetLocal ());
2407         // std::cout << "Broadcast RERR message from interface " << i->GetLocal () << std::endl;
2408         // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
2409         Ptr<Packet> p = packet->Copy ();
2410         Ipv4Address destination;
2411         if (i->GetMask () == Ipv4Mask::GetOnes ())
2412           {
2413             destination = Ipv4Address ("255.255.255.255");
2414           }
2415         else
2416           {
```

```
2417            destination = i->GetBroadcast ();
2418          }
2419        Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
                ↪ &RoutingProtocol::SendTo, this, socket, p, destination);
2420      }
2421    }
2422  }
2423
2424  Ptr<Socket>
2425  RoutingProtocol::FindSocketWithInterfaceAddress (Ipv4InterfaceAddress addr ) const
2426  {
2427    NS_LOG_FUNCTION (this << addr);
2428    for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
2429          m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
2430      {
2431        Ptr<Socket> socket = j->first;
2432        Ipv4InterfaceAddress iface = j->second;
2433        if (iface == addr)
2434          {
2435            return socket;
2436          }
2437      }
2438    Ptr<Socket> socket;
2439    return socket;
2440  }
2441
2442  Ptr<Socket>
2443  RoutingProtocol::FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress addr ) const
2444  {
2445    NS_LOG_FUNCTION (this << addr);
2446    for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
2447          m_socketSubnetBroadcastAddresses.begin (); j != m_socketSubnetBroadcastAddresses.end (); ++j)
2448      {
2449        Ptr<Socket> socket = j->first;
2450        Ipv4InterfaceAddress iface = j->second;
2451        if (iface == addr)
2452          {
2453            return socket;
2454          }
2455      }
2456    Ptr<Socket> socket;
2457    return socket;
2458  }
2459
2460  void
2461  RoutingProtocol::DoInitialize (void)
2462  {
2463    NS_LOG_FUNCTION (this);
2464    uint32_t startTime;
2465    if (m_enableHello)
2466      {
2467        m_htimer.SetFunction (&RoutingProtocol::HelloTimerExpire, this);
2468        startTime = m_uniformRandomVariable->GetInteger (0, 100);
2469        NS_LOG_DEBUG ("Starting␣at␣time␣" << startTime << "ms");
2470        m_htimer.Schedule (MilliSeconds (startTime));
2471      }
2472    Ipv4RoutingProtocol::DoInitialize ();
2473  }
2474
2475  void
2476  RoutingProtocol::getCoordinates(void)
2477  {
2478     Ptr<MobilityModel> mobNode = m_ipv4->GetObject<Node>()->GetObject<MobilityModel>();
2479     this->nodeX = mobNode->GetPosition().x;
2480     this->nodeY = mobNode->GetPosition().y;
2481     return;
2482  }
2483
```

```
void
RoutingProtocol::DecayActiveEnergy(uint32_t oldValue, uint32_t newValue) {
  std::cout<<"Application␣triggered␣active␣energy␣decay␣with␣tick␣value:␣"<<newValue<<std::endl;
}

float
RoutingProtocol::GetDistanceBetweenNodes(RoutingTableEntry route)
{
    // Calculates hypotenuse from current and next node coordinates
    float a_sq = powf(abs(nodeX - route.GetNextHopXLoc()), 2);
    float b_sq = powf(abs(nodeY - route.GetNextHopYLoc()), 2);
    return sqrt(a_sq + b_sq);
}

} //namespace lsaodv
} //namespace ns3
```

LS-AODV simulation application

lsAodvSimulation.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * 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
 *
 * This is an example script for EA_AODV manet routing protocol.
 *
 * Authors: Pavel Boyko <boyko@iitp.ru>
 */

#include <iostream>
#include <cmath>
#include "ns3/ls-aodv-module.h"
#include "ns3/aodv-module.h"
#include "ns3/ea-aodv-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/v4pingRand-helper.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/vector.h"
#include <ctime>
#include <array>
#include <random>
using namespace ns3;

#define NUM_NODES 50
#define NODE_POSE_MEAN 125
#define NODE_POSE_STD_DEV 50
```

```
#define NODE_POSE_MAX 250

#define TRANS_DISTANCE 250

std::array <Vector,NUM_NODES> nodePositions;

class SimCompare
{
public:
  SimCompare ();
  virtual ~SimCompare ();
  /**
   * \brief Configure script parameters
   * \param argc is the command line argument count
   * \param argv is the command line arguments
   * \return true on successful configuration
  */
  bool Configure (int argc, char **argv);
  /// Run simulation
  void Run ();
  /**
   * Report results
   * \param os the output stream
   */
  void Report (std::ostream & os);


  // parameters
  /// Number of nodes
  uint32_t size;
  /// Distance between nodes, meters
  double step;
  /// Simulation time, seconds
  double totalTime;
  /// Write per-device PCAP traces if true
  bool pcap;
  /// Print routes if true
  bool printRoutes;
public:
  // network
  /// nodes used in the example
  NodeContainer nodes;
  /// devices used in the example
  NetDeviceContainer devices;
  /// interfaces used in the example
  Ipv4InterfaceContainer interfaces;

  /// Create the nodes
  void CreateNodes ();
  /// Create the devices
  void CreateDevices ();
  /// Create the network
  virtual void InstallInternetStack ();
  /// Create the simulation applications
  void InstallApplications ();
  void CleanUp ();
};

class AodvSim : public SimCompare {
  public:
    AodvSim();
    virtual ~AodvSim();
    void InstallInternetStack ();
};

class EaAodvSim : public SimCompare {
  public:
    EaAodvSim();
```

```
112     virtual ~EaAodvSim();
113     void InstallInternetStack ();
114 };
115 
116 
117 class LsAodvSim : public SimCompare {
118   public:
119     LsAodvSim();
120     virtual ~LsAodvSim();
121     void InstallInternetStack ();
122 };
123 
124 int main (int argc, char **argv)
125 {
126   // create random normal distribution for node positions to be use by all
127   // mobility models
128   std::default_random_engine generator;
129   generator.seed(12346);
130   std::normal_distribution<double> distribution(NODE_POSE_MEAN,NODE_POSE_STD_DEV);
131 
132   for (Vector& i : nodePositions) {
133     do {i.x = distribution(generator);}
134     while (i.x < 0 || i.x > NODE_POSE_MAX);
135     do {i.y = distribution(generator);}
136     while (i.y < 0 || i.y > NODE_POSE_MAX);
137     i.z = 0.0;
138     //std::cout<<"("<<i.x<<","<<i.y<<","<<i.z<<")"<<std::endl;
139   }
140 
141   // run aodv simulation
142   AodvSim aodv;
143   std::cout<<"--------␣AODV␣SIMULATION␣----------"<<std::endl;
144   if (!aodv.Configure (argc, argv))
145     NS_FATAL_ERROR ("Configuration␣failed.␣Aborted.");
146 
147   aodv.Run ();
148   aodv.Report (std::cout);
149   std::cout<<"-----------------------------------"<<std::endl<<std::endl;
150 
151   // run ea-aodv simulation
152   EaAodvSim eaaodv;
153   std::cout<<"--------␣EA-AODV␣SIMULATION␣----------"<<std::endl;
154   if (!eaaodv.Configure (argc, argv))
155     NS_FATAL_ERROR ("Configuration␣failed.␣Aborted.");
156 
157   eaaodv.Run ();
158   eaaodv.Report (std::cout);
159   std::cout<<"-----------------------------------"<<std::endl<<std::endl;
160 
161 
162   // run ls-aodv simulation
163   LsAodvSim lsAodv;
164   std::cout<<"-------␣LS-AODV␣SIMULATION␣--------"<<std::endl;
165   if (!lsAodv.Configure (argc, argv))
166     NS_FATAL_ERROR ("Configuration␣failed.␣Aborted.");
167 
168   lsAodv.Run ();
169   lsAodv.Report (std::cout);
170   std::cout<<"-----------------------------------"<<std::endl<<std::endl;
171   return 0;
172 }
173 
174 //-----------------------------------------------------------------------------
175 SimCompare::SimCompare () :
176   // number of nodes
177   size (NUM_NODES),
178   step (100),
179   // run time
```

```
  totalTime (10),
  pcap (false),
  printRoutes (false)
{
}

SimCompare::~SimCompare() {}

AodvSim::AodvSim() {}
AodvSim::~AodvSim() {
  Names::Clear();
}

EaAodvSim::EaAodvSim(){}
EaAodvSim::~EaAodvSim() {
  Names::Clear();
}

LsAodvSim::LsAodvSim(){}
LsAodvSim::~LsAodvSim() {
  Names::Clear();
}

bool
SimCompare::Configure (int argc, char **argv)
{
  // Enable EA_AODV logs by default. Comment this if too noisy
  // LogComponentEnable("AodvRoutingProtocol", LOG_LEVEL_ALL);

  SeedManager::SetSeed (12345);
  CommandLine cmd;

  cmd.AddValue ("pcap", "Write␣PCAP␣traces.", pcap);
  cmd.AddValue ("printRoutes", "Print␣routing␣table␣dumps.", printRoutes);
  cmd.AddValue ("size", "Number␣of␣nodes.", size);
  cmd.AddValue ("time", "Simulation␣time,␣s.", totalTime);
  cmd.AddValue ("step", "Grid␣step,␣m", step);

  cmd.Parse (argc, argv);
  return true;
}

void
SimCompare::Run ()
{
// Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", UintegerValue (1)); // enable rts
    ↪ cts all the time.
  CreateNodes ();
  CreateDevices ();
  InstallInternetStack ();
  InstallApplications ();

  std::cout << "Starting␣simulation␣for␣" << totalTime << "␣s␣...\n";

  Simulator::Stop (Seconds (totalTime));
  Simulator::Run ();
  Simulator::Destroy ();
  CleanUp();
}

void
SimCompare::Report (std::ostream &)
{
}

void
SimCompare::CreateNodes ()
{
```

```
247   std::cout << "Creating " << (unsigned)size << " nodes.\n";
248   nodes.Create (size);
249   // Name nodes
250   for (uint32_t i = 0; i < size; ++i)
251     {
252       std::ostringstream os;
253       os << "node-" << i;
254       Names::Add (os.str (), nodes.Get (i));
255     }
256   // Create random topology
257   MobilityHelper mobility;
258   /*mobility.SetPositionAllocator("ns3::GridPositionAllocator",
259                                 "MinX",DoubleValue(0),
260                                 "MinY",DoubleValue(0),
261                                 "DeltaX",DoubleValue(50),
262                                 "DeltaY",DoubleValue(50),
263                                 "GridWidth",UintegerValue((int)sqrt(size)),
264                                 "LayoutType",StringValue("RowFirst"));*/
265   // area of network
266   //mobility.SetPositionAllocator("ns3::RandomRectanglePositionAllocator",
267   //               "X",StringValue("ns3::NormalRandomVariable[Mean=125|Variance=2500|Bound=250]"),
268   //               "Y",StringValue("ns3::NormalRandomVariable[Mean=125|Variance=2500|Bound=250]"));
269   /*mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
270                                  "MinX", DoubleValue (0.0),
271                                  "MinY", DoubleValue (0.0),
272                                  "DeltaX", DoubleValue (step),
273                                  "DeltaY", DoubleValue (0),
274                                  "GridWidth", UintegerValue (size),
275                                  "LayoutType", StringValue ("RowFirst"));*/
276   mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
277   mobility.Install (nodes);
278   //for (auto&& [i,pos] = std::tuple{0,nodePositions.data()} ; i<NUM_NODES ; pos++,i++) {
279   // Config::Set("/NodeList/"+std::to_string(i)+"/$ns3::MobilityModel/Position",
280   //  Vector3DValue(*pos));
281   //}
282   for (int i = 0; i < NUM_NODES; i++) {
283     Config::Set("/NodeList/"+std::to_string(i)+"/$ns3::MobilityModel/Position",
284       VectorValue(nodePositions[i]));
285   }
286   std::cout<<"Assigned mobility model for nodes"<<std::endl;
287 }
288
289 void
290 SimCompare::CreateDevices ()
291 {
292   WifiMacHelper wifiMac;
293   wifiMac.SetType ("ns3::AdhocWifiMac");
294   YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
295   YansWifiChannelHelper wifiChannel; /*= YansWifiChannelHelper::Default ();*/
296   wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
297   wifiChannel.AddPropagationLoss ("ns3::RangePropagationLossModel");
298   Config::SetDefault("ns3::RangePropagationLossModel::MaxRange",DoubleValue(250.0));
299   wifiPhy.SetChannel (wifiChannel.Create ());
300   WifiHelper wifi;
301   wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode", StringValue ("OfdmRate6Mbps"),
        ↪ "RtsCtsThreshold", UintegerValue (0));
302   devices = wifi.Install (wifiPhy, wifiMac, nodes);
303
304   if (pcap)
305     {
306       wifiPhy.EnablePcapAll (std::string ("lsaodv"));
307     }
308 }
309
310 void SimCompare::InstallInternetStack() {}
311
312 void
313 LsAodvSim::InstallInternetStack ()
```

```
{
  lsaodvHelper lsAodv;
  // you can configure EA_AODV attributes here using ea-aodv.Set(name, value)
  lsAodv.Set("EnableHello",BooleanValue(false));
  InternetStackHelper stack;
  stack.SetRoutingHelper (lsAodv); // has effect on the next Install ()
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.0.0.0", "255.0.0.0");
  interfaces = address.Assign (devices);
  // all nodes start with 1000 joules of energy
  Config::SetDefault("ns3::lsaodv::RoutingProtocol::TransDistance",DoubleValue(250.0));
  // energy level of all nodes
  Config::Set("/NodeList/*/$ns3::lsaodv::RoutingProtocol/MaxPower",DoubleValue(100));
  //Config::Set("/NodeList/*/$ns3::lsaodv::RoutingProtocol/MaxPower",DoubleValue(1e16));

  if (printRoutes)
    {
      Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("lsAodv.routes", std::ios::out);
      lsAodv.PrintRoutingTableAllAt (Seconds (totalTime - 0.001), routingStream);
    }
}

void
EaAodvSim::InstallInternetStack ()
{
  EaAodvHelper eaAodv;
  // you can configure EA_AODV attributes here using ea-aodv.Set(name, value)
  eaAodv.Set("EnableHello",BooleanValue(false));
  InternetStackHelper stack;
  stack.SetRoutingHelper (eaAodv); // has effect on the next Install ()
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.0.0.0", "255.0.0.0");
  interfaces = address.Assign (devices);
  // all nodes start with 1000 joules of energy
  Config::SetDefault("ns3::eaAodv::RoutingProtocol::TransDistance",DoubleValue(250.0));
  // energy level of all nodes
  Config::Set("/NodeList/*/$ns3::eaAodv::RoutingProtocol/MaxPower",DoubleValue(100));
  //Config::Set("/NodeList/*/$ns3::eaAodv::RoutingProtocol/MaxPower",DoubleValue(1e16));

  if (printRoutes)
    {
      Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("eaAodv.routes", std::ios::out);
      eaAodv.PrintRoutingTableAllAt (Seconds (totalTime - 0.001), routingStream);
    }
}


void
AodvSim::InstallInternetStack ()
{
  AodvHelper Aodv;
  // you can configure EA_AODV attributes here using ea-aodv.Set(name, value)
  Aodv.Set("EnableHello",BooleanValue(false));
  InternetStackHelper stack;
  stack.SetRoutingHelper (Aodv); // has effect on the next Install ()
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.0.0.0", "255.0.0.0");
  interfaces = address.Assign (devices);
  // all nodes start with 1000 joules of energy
  Config::SetDefault("ns3::aodv::RoutingProtocol::TransDistance",DoubleValue(250.0));
  // energy level of all nodes
  Config::Set("/NodeList/*/$ns3::aodv::RoutingProtocol/MaxPower",DoubleValue(100));

  if (printRoutes)
    {
```

```
382       Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("Aodv.routes", std::ios::out);
383       Aodv.PrintRoutingTableAllAt (Seconds (totalTime - 0.001), routingStream);
384     }
385 }
386 
387 void SimCompare::InstallApplications ()
388 {
389   V4PingRandHelper ping (interfaces.GetAddress (size - 1));
390   ping.SetAttribute ("Verbose", BooleanValue (true));
391   ping.SetAttribute ("Network",Ipv4AddressValue("10.0.0.0"));
392   ping.SetAttribute ("NetSize", UintegerValue(size));
393   // vary reciprocal of injection rate
394   ping.SetAttribute("Interval",TimeValue(MilliSeconds(1000)));
395   //ApplicationContainer p = ping.Install (nodes.Get (0));
396   for (uint32_t i = 0; i < size; i++) {
397     ApplicationContainer p = ping.Install (nodes.Get (i));
398     Ptr<Node> currNode = nodes.Get(i);
399     // Get node address for v4Ping application
400     //Ptr<Ipv4> ipv4 = currNode->GetObject<Ipv4>();
401     //Ipv4InterfaceAddress ipv4iAddr = ipv4->GetAddress(0,0);
402     //Ipv4Address ipv4Addr = ipv4iAddr.GetLocal();
403     Ipv4Address ipv4Addr = interfaces.GetAddress(i);
404     //std::cout<<"Node address "<<ipv4Addr<<std::endl;
405     std::string cmd = "/NodeList/";
406     cmd+=std::to_string(i);
407     cmd+="/ApplicationList/0/$ns3::V4PingRand/MyAddr";
408     Config::Set(cmd,Ipv4AddressValue(ipv4Addr));
409     p.Start (Seconds (0));
410     p.Stop (Seconds (totalTime) - Seconds (0.001));
411   }
412 
413   // move node away
414   //Ptr<Node> node = nodes.Get (size/2);
415   //Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
416   //Simulator::Schedule (Seconds (totalTime/3), &MobilityModel::SetPosition, mob, Vector (1e5, 1e5, 1e5));
417 }
418 
419 void SimCompare::CleanUp() {
420   Names::Clear();
421 }
```

References

[1] Das, S.R., Perkins, C.E., Royer, E.M.: *Ad hoc on demand distance vector routing.* In: IETF Internet Draft (2001).

[2] P. Nayak, R. Agarwal and S. Verma,"Energy Aware AODV (EA-AODV) Using Variable Range Transmission," Advances in Computer Science, Engineering & Applications, pp. 589-597, 2012.

[3] "Humidity and Temperature Sensor Node for Star Networks Enabling 10+ Year Coin Cell Battery Life," Texax Instruments Incorporated, May, 2016.

[4] Texas Instruments, "CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee® Applications," CC2538 datasheet, April, 2015.

[5] Texas Instruments, "TLV62150x 4-V to 17-V 1-A Step-Down Converter In 3x3 QFN Package," TLV62150 datasheet, September 2016.

[6] Texas Instruments, "TPS56120x 4.5-V to 17-V Input, 1-A Synchronous Step-Down Voltage Regulator in 6-Pin SOT-23," TPS561201 datasheet, September 2020.

[7] Texas Instruments, "CC2592 2.4-GHz Range Extender," CC2592 datasheet, February 2014.

[8] Bosch Sensortec, "BMP280 Digital Pressure Sensor," BMP280 datasheet, November 2020.

[9] Alexander R. et al., *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.* In: IETF Internet Draft (2012).

[10] Linear Technology, "Minimizing Switching Regulator Residue in Linear Regulator Outputs," Appl. Note 101, July 2005.

[11] Analog Devices, "Capacitor Selection Guidelines for Analog Devices, Inc., LDOs," Appl. Note 1099, 2010.

[12] Lee, Bang S., "Understanding the Terms and Definitions of LDO Voltage Regulators," Texas Instruments, Application Report, October 1999.

[13] Texas Instruments, "TPS7A05 1-µA Ultralow IQ, 200-mA, Low-Dropout Regulator in a Small-Size Package," TPS7A05 datasheet, August 2019.

[14] Texas Instruments, "TPS7A20 300-mA, Ultra-Low-Noise, Low-IQ, High PSRR LDO," TPS7A20 datasheet, July 2021.

[15] Adesto, "AT25SF321B 32-Mbit SPI Serial Flash Memory with Dial-I/O and Quad-I/O Support," AT25SF321B datasheet, May 2020.

[16] "ns-3 Tutorial – Tutorial," NSNAM. Accessed: Nov. 14, 2020. [Online]. Available: https://www.nsnam.org/docs/tutorial/html/.

[17] F. Hu, X. Cao, *Wireless Sensor Networks, Principles and Practice*, 1st ed. Boca Raton, FL: Auerbach Publications, 2010.

[18] Espressif Systems, "ESP32 Series Datasheet," ESP32 Datasheet, 2020 [Revised April 2020].

[19] T. Rappaport, *Wireless Communications, Principles and Practice*, 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR, 2002, p. 107.

[20] "Supplement to IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band," IEEE Std 802.11b, 1999.