An Energy Framework for the Network Simulator 3 (ns-3)

He Wu, Sidharth Nabar and Radha Poovendran
Network Security Lab (NSL)
Electrical Engineering Department
University of Washington, Seattle, USA
mdzz@uw.edu, snabar@uw.edu, rp3@uw.edu

ABSTRACT

The Network Simulator-3 (ns-3) is rapidly developing into a flexible and easy-to-use tool suitable for wireless network simulation. Since energy consumption is a key issue for wireless devices, wireless network researchers often need to investigate the energy consumption at a battery powered node or in the overall network, while running network simulations. This requires the underlying simulator to support energy consumption and energy source modeling. Currently however, ns-3 does not provide any support for modeling energy consumption or energy sources. In this paper, we introduce an integrated energy framework for ns-3, with models for energy source as well as energy consumption. We present the design and implementation of the overall framework and the specific models therein. Further, we show how the proposed framework can be used in ns-3 to simulate energy-aware protocols in a wireless network.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development—*modeling methodologies*

General Terms

Design, Algorithm, Performance, Verification

Keywords

ns-3, energy consumption, energy-aware protocol simulation, network simulations, wireless networks, battery models

1. INTRODUCTION

Network simulators are widely used in wireless network research to provide a controlled environment for researchers to perform experimental evaluation of protocols when hardware resources are limited. Simulations are used as the first step to validate new protocols as well as to compare their performance to that of existing, standard protocols. In this paper, we focus on the Network Simulator 3 (ns-3) [2], which is an open source network simulator intended to replace the Network Simulator 2 (ns-2) [1]. ns-3 is written in C++ with a highly flexible architecture. It allows third party contributors to design new models and incorporate them into the main ns-3 code, resulting in a continually increasing scope. Components such as WiFi and WiMAX standard implementations, mobility models and routing protocols, make ns-3 well suited for wireless network simulations.

Since wireless network nodes are typically powered by batteries, the amount of energy available at each node, and hence in the overall network, is limited. As a result, utilization of energy is an important performance metric for wireless network operation. In order to allow modeling of energy as a network resource in simulation studies, a simulator should provide: a) models for energy consumption or expenditure; and b) models for energy sources. Models for energy consumption capture how energy is consumed at a network node. In order to support a wide range of network simulations, it is desirable to include device energy consumption models for various components of a node, such as radios, processors or sensors.

Energy source models represent the power supplies or batteries of network nodes, and include linear as well as non-linear battery models. Linear, ideal source models are easy to set up and configure, but they do not capture crucial aspects of real-life batteries, such as the dependency of battery efficiency on load current [8]. On the other hand, non-linear battery models are more difficult to configure, but are typically more accurate since they model the discharge curves of specific batteries. Further, new ways of providing energy such as fuel cells and energy scavenging are becoming viable for use in wireless networks [9]. Energy scavenging is the process of acquiring and storing energy from the device's operating environment. It provides longer lifetime to small wireless devices without the need for a large battery. In order to include these new types of energy sources, new energy source models may need to be defined.

A network simulator should allow users to choose between multiple types of energy sources, and use either one in combination with the aforementioned set of device energy consumption models. In order to ensure easy usage and compatibility between existing and future source and device models, it is necessary to integrate individual device energy consumption and energy source models into a unified *Energy Framework*. Such a framework will allow investigation of energy consumption at a node as well as overall network

SIMUTools 2011 March 21–25, Barcelona, Spain.

lifetime under specific conditions (for example, when using a specific routing protocol). Currently however, ns-3 does not have any energy framework nor the ability to incorporate energy into network simulations.

In this paper, we report a contribution[1] to ns-3 comprising of an energy framework. This framework adds sufficient support to ns-3 to allow simulation of energy consumption in wireless networks. Our contributions include:

- Energy source models for ideal, linear energy sources as well as for non-linear batteries.
- Device energy models that represent energy consumption of devices such as the WiFi radio.
- Methods that provide energy consumption information to ns-3 objects external to the framework. These methods allow users to simulate energy-aware protocols that adjust their operation based on information about energy consumption at a node.
- An integrated framework that ensures compatibility between different energy source and device energy consumption models and allows easy integration of new models.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 presents the design goals and overall structure of the proposed ns-3 energy framework. We then present in detail the design and implementation of the framework in Section 4. In Section 5 we discuss the validation of the proposed energy framework and demonstrate the framework's usage in wireless network simulations. Section 6 discusses possible directions of extending the proposed energy framework. Section 7 concludes the paper.

2. RELATED WORK

ns-3 is an open source discrete event simulator capable of performing simulations involving complex network topologies, virtual networks and network testbeds. Features such as the attribute system, automatic memory management and configurable tracing system [2] make ns-3 very easy to use for a wide range of network simulations. As an open-source software, ns-3 has great flexibility for researchers to develop and contribute new models. As noted in Section 1, however, ns-3 is currently unable to support simulations involving energy consumption, due to lack of energy source and consumption models.

In terms of energy consumption modeling, existing network simulators focus on modeling radio energy consumption [3] because the radio is assumed to consume the most power on a wireless node. These radio energy consumption models allow users to specify power consumption of the radio for different operating states. These values are then multiplied by the time spent by the radio in the respective states to calculate energy consumption. ns-2 [1] uses this approach to model radio energy consumption and does not have a separate module for energy sources.

For modeling energy sources, common approaches in literature are: ideal, linear energy source and non-linear battery model. The ideal, linear energy source has a user-specified capacity in *Joules* and discharges linearly towards zero. Such a model is easy to setup, since only the capacity and supply voltage parameters need to be specified [1]. However, real-life batteries exhibit nonlinear effects, namely *Rate Capacity Effect* and *Recovery Effect* [8]. The *Rate Capacity Effect* is the decrease in battery lifetime when the current draw is higher then the rated value of the battery. The *Recovery Effect* is the increase in battery lifetime when the battery is alternating between discharge and idle states. Ignoring these effects in battery models may lead to incorrect simulation conclusions. For example, using the ideal, linear energy model in simulation implies that switching the radio between operating and idle states leads to energy consumption. However, as shown in [13], this is not true when *Recovery Effect* is considered. Also, the reduced battery life caused by *Rate Capacity Effect* is not captured by the ideal, linear battery model. Therefore, it is important for battery models to capture both *Rate Capacity Effect* and *Recovery Effect*.

These effects are captured by several nonlinear models proposed in literature [8]. These are classified as electrochemical, stochastic, electrical circuit and analytical models. Electrochemical models [8] are most accurate but computationally intensive; stochastic models [8] require building of look up tables for battery non-linearity, which makes them cumbersome to use; and circuit models [8] use circuit simulation tools which are difficult to integrate with network simulation tools. Therefore, these models are not suitable for integration with ns-3.

Finally, analytical models [10] use battery discharge curves to construct mathematical expressions for battery behavior. Such models are accurate, computationally efficient and easy to configure. Specifically, the Rakhmatov-Vrudhula (R-V) model introduced in [10, 11] is able to characterize a battery by specifying only two parameters. However, this model requires prior knowledge of the complete load profile of the battery, making integration with network simulators difficult. To address this problem, a runtime version of the R-V model was proposed in [7]. We adopt and modify this algorithm for a battery model implementation in the proposed ns-3 energy framework.

3. ENERGY FRAMEWORK FOR NS-3

The basic design goals, assumptions and the overall structure of the proposed framework are presented in this section.

3.1 Design Goals

In order to ensure that the proposed energy framework can be easily configured, used and extended for a wide range of applications, the following design goals are identified:

- *Wide Scope*: The energy framework must be able to support a wide range of functionalities that users of ns-3 may require in energy-related simulation. This includes models for energy consumption, realistic battery discharge and energy scavenging.
- *Modular Design*: Design of the energy framework must be highly modular in order to ensure that different parts of the framework have minimal interdependencies. This enables users to easily conduct simulation using various combination of models, or to modify an existing model with minimal effect on other, connected models.

[1]The proposed energy framework is included in ns-3.9, released in Aug 2010. Source code is available in the `src/contrib/energy` folder of ns-3.9.

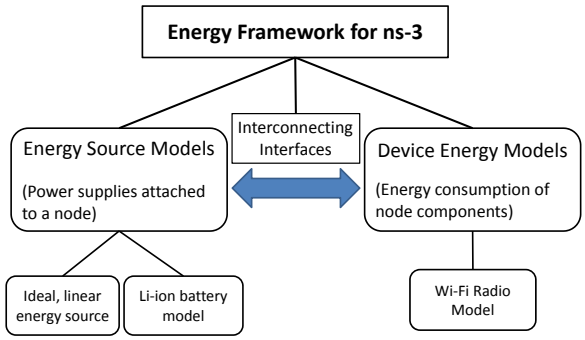


Figure 1: Proposed ns-3 Energy Framework Structure, including energy sources, device energy models and the interfaces between them.

- *Framework Extensibility*: As techniques in energy consumption and energy source modeling advance, new device or source models may be created to integrate with the energy framework. A highly extensible framework enables design, integration and use of new source or device models with minimal modifications to the basic framework structure.

- *Ease of Use and Low Simulation Overhead*: The models included in the energy framework must be easy to configure in ns-3 simulations. Also, adding the energy framework should have minimal impact on the overall code size of the simulation and the simulation speed.

3.2 Assumptions

The proposed ns-3 energy framework is based on the following specific assumptions:

- A network node can be powered either by a single energy source or by a set of energy sources. In the case of multiple sources, these are assumed to be independent and cannot perform intelligent load sharing among themselves.

- Energy sources supply power to devices on the node at a constant voltage. In real devices, this is achieved through the use of voltage regulators in the power supply circuit.

- Operation of all devices on the node is state-based. Each state of the device's operation has a specific load current value associated with it. For example, a radio could have four states defined as *Receive, Transmit, Idle* and *Sleep*, each of which is associated with a corresponding current draw value.

3.3 Basic Structure of Proposed Framework

As shown in Figure 1, the proposed ns-3 energy framework consists of a set of energy sources, a set of device energy models, and the interfaces interconnecting them. Energy source models represent different types of energy sources, such as Li-ion batteries. Device energy models represent components of a node, for example, a WiFi radio, which consume energy from the energy source.

Energy sources are used to power multiple devices and energy consumption of each device is represented by its corresponding device energy model. The interactions between an energy source and device energy models are twofold: device energy models consume energy from the energy source, and the energy source notifies device energy models when its energy capacity is completely drained. The proposed ns-3 energy framework also provides querying functions to external simulation modules. For example, an energy aware routing protocol may request information about the battery level at the node, which can be obtained by querying the energy source on the node.

The structure described above is designed to represent most real-life scenarios where energy is supplied to a node by a certain type of energy source and multiple components of the node consume energy from this source. We separate the energy providing process from the energy consuming process in the framework so that they can be modeled individually, thus making the framework highly modular. Further, defining a set of basic, standard interfaces provides flexibility to users, allowing them to use any energy source and device model combination suitable for the simulation while ensuring compatibility between models.

4. IMPLEMENTATION DETAILS

In this section, we first review some key features of ns-3 utilized for our implementation, followed by a description of the various modules included in our framework. We define the *EnergySource* and *DeviceEnergyModel* classes to represent energy sources and device energy models respectively. The detailed structure of the proposed framework is shown in Figure 2.

4.1 Review of ns-3 Features

ns-3 extends the basic C++ object system by introducing the ns-3 *Object* class to provide additional functionalities and features such as the attribute system and object aggregation [2]. For the remainder of this paper, we use the term "object" to refer to the ns-3 *Object* class.

The basic entity of a ns-3 network topology is represented by a *Node* object. Individual components of the node are usually defined as objects and are attached to the node. For example, in a simulation, a laptop can be represented as a node and a component such as the WiFi radio would be installed on it as an object. ns-3 uses object aggregation to install or attach an object onto a node. Object aggregation allows users to attach different objects onto a node at runtime, to provide additional functionality without having to modify the node class. However, only one object of a certain class can be aggregated to a node. In our energy framework, object aggregation is used to attach a container of *EnergySource* objects onto a node.

Another important feature of ns-3 is the attribute system. Generally, in order to modify certain parameters of a simulation, users need to access the underlying objects and their internal member variables, which involves modifications to source code of specific classes. Setting member variables of an object as attributes allows users to easily specify the values of such variables at runtime in the high-level simulation script. In our energy framework, model parameters such as the initial energy and the supply voltage of an *EnergySource* are defined as attributes.

Finally, simulation events in ns-3 are managed and scheduled by the *Simulator* class. In the proposed energy framework, we use the *Simulator* class to schedule periodic events such as the regular update of the remaining energy.

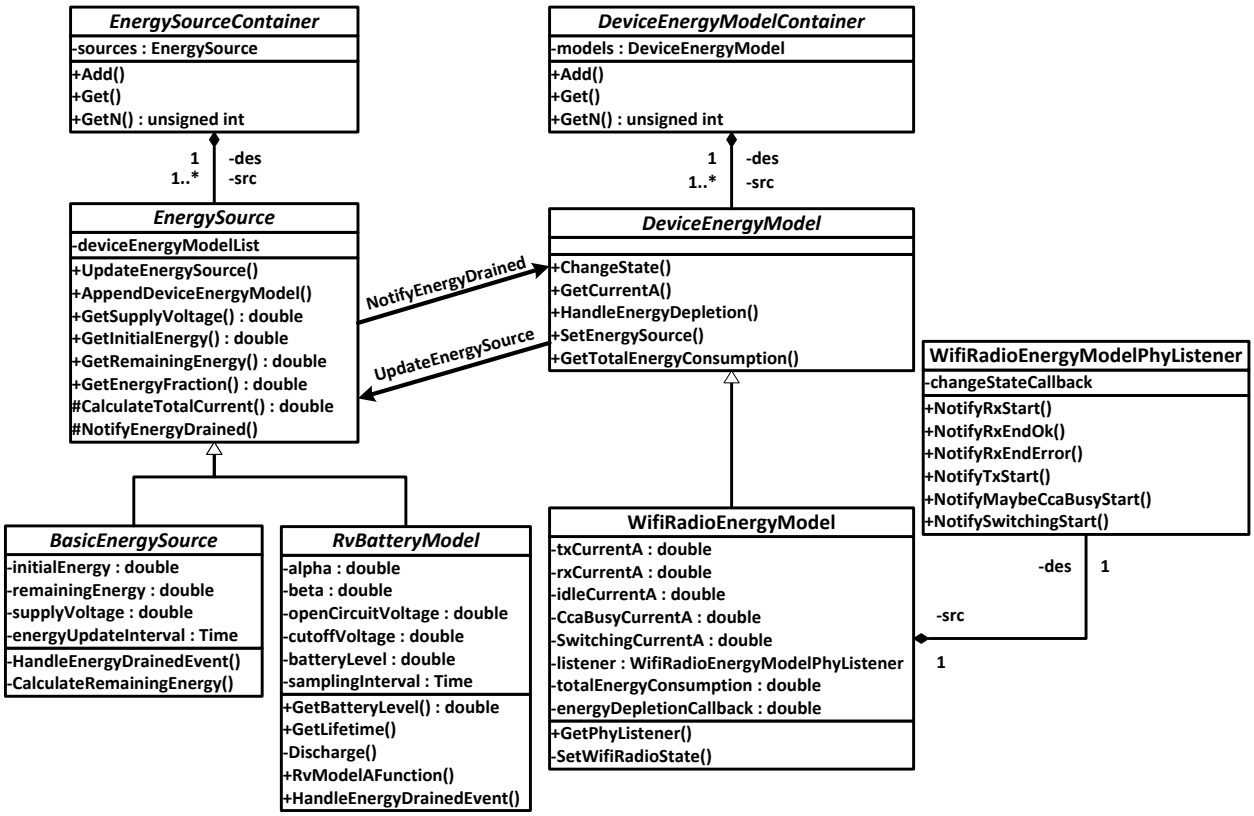


Figure 2: Class diagram of the proposed ns-3 energy framework, including *EnergySource*, *DeviceEnergyModel* and child classes.

4.2 Energy Source Class

The *EnergySource* class represents the power supply of a network node. One or more *EnergySource* objects are packaged into a single *EnergySourceContainer* object, which is then attached onto a node by means of aggregation. Each *EnergySource* can be used to power a set of components on the node which are maintained as a list of *DeviceEnergyModel* objects within the *EnergySource*. The *EnergySource* class is defined as an abstract base class and its child classes are characterized by how they model the discharging of the source.

The overall operation of the *EnergySource* is similar for all child implementations: At the beginning of a ns-3 simulation, the *EnergySource* on each node contains a certain amount of initial energy specified by user. During simulation, energy will be consumed by *DeviceEnergyModel* objects connected to the *EnergySource*. Finally if energy is completely drained, the *EnergySource* sends a notification to all *DeviceEnergyModel* objects connected to it. Each *DeviceEnergyModel* object then handles the notification independently.

Since child classes of *EnergySource* include linear as well as nonlinear battery models, its interfaces must be applicable for both these types of models. For ideal, linear energy source, energy consumption can be calculated using either power values or current draw values from the *DeviceEnergyModel*. For nonlinear sources, in order to account for the *Rate Capacity Effect*, total current draw value from all devices attached to the *EnergySource* is needed [10, 11, 7]. Therefore, current draw value is selected to represent the load to the *EnergySource*. The total current draw is calculated as the sum of current draw from all *DeviceEnergyModel* objects attached to the same *EnergySource*.

We define the *EnergySource* API as follows:

- *GetInitialEnergy*: Returns the initial energy stored in the energy source.
- *GetRemainingEnergy*: Returns the remaining energy in the energy source.
- *GetEnergyFraction*: Returns the energy fraction of *EnergySource*. Energy fraction is defined here as (*Remaining Energy* ÷ *Initial Energy*).
- *UpdateEnergySource*: Used by *DeviceEnergyModel* objects to notify the *EnergySource* to update its remaining energy when a state change has occurred in the devices. The *EnergySource* can also schedule periodic calls of this function to constantly update the remaining energy values. The exact method used for the update depends on the specific energy source model being implemented.
- *AppendDeviceEnergyModel*: Used to add a *DeviceEnergyModel* to the list of *DeviceEnergyModel* objects kept within *EnergySource* object.
- *CalculateTotalCurrent*: This protected method queries all *DeviceEnergyModel* objects attached to the *EnergySource* for their current draw and then calculates the total current drawn from the energy source.

- *NotifyEnergyDrained*: This protected method is called when energy is completely drained from the *EnergySource* and sends a notification to all *DeviceEnergyModel* objects attached to the *EnergySource*.

Two child classes are implemented for the *EnergySource* base class: *BasicEnergySource* and *RvBatteryModel*. The *BasicEnergySource* is an ideal, linear energy source similar to the one included in ns-2 [1]. The *RvBatteryModel* is a nonlinear battery model capable of modeling both *Rate Capacity Effect* and *Recovery Effect*.

4.2.1 Basic Energy Source Class

The *BasicEnergySource* stores an initial amount of energy E_0 defined by user and this reservoir of energy is continually consumed by *DeviceEnergyModel* objects. It also stores the i^{th} sample of the total current draw at the node (I_i) and the time stamp associated with it (t_i). When a device on a node changes its operating state, the *BasicEnergySource* is notified. The *BasicEnergySource* then calculates and stores the new load value (I_{i+1}) and the time stamp (t_{i+1}). The *BasicEnergySource* class calculates the energy consumption during the period (t_i, t_{i+1}) as follows:

$$E_{i+1} = E_i + V \times (t_{i+1} - t_i) \times I_i \quad (1)$$

where E_i is energy consumption of the source at t_i, and V is the supply voltage.

The update of remaining energy is also driven from within the *BasicEnergySource*. Periodic calls to the *UpdateEnergySource* function are scheduled with a user defined interval so that when constant load is applied, *BasicEnergySource* can still keep track of energy reduction. When external objects requests for energy information from the *BasicEnergySource*, for example, through *GetRemainingEnergy*, the *UpdateEnergySource* function is always invoked so that the latest energy information will be returned.

4.2.2 R-V Battery Model Class

The *RvBatteryModel* class implements the Rakhmatov-Vrudhula (R-V) battery model [10, 11] which characterizes a battery using two parameters: α and β. The parameter α measures the capacity of the battery in *Coulombs* while the parameter β measures the battery nonlinearity. β^2 is measured in units of $1/second$. These parameters can be obtained using discharge curves from battery data sheets or from simple constant load experiments [11].

A runtime algorithm of the R-V battery model is proposed in [7], where the current draw is sampled periodically as shown in Figure 3a. The current draw sample and the associated time stamp or the load profile is then stored by the `discharge` function. At each sampling instance, the `discharge` method calculates an estimate of the battery's α parameter, denoted by α_{est}, using the recorded load profile. α_{est} indicates how much of the battery's capacity has been discharged. When α_{est} exceeds the battery's α parameter, the algorithm declares the battery as completely drained and the current time is set as the battery lifetime. Details of the `discharge` function can be found in [7].

Implementation of the R-V battery model in the proposed energy framework uses a modified version of the above mentioned runtime algorithm. As shown in Figure 3a, the algorithm in [7] samples the current draw at a user specified interval of Δt starting at t_0. It can be observed that the current draw I_1 is omitted as I_1's interval is smaller then the sampling interval Δt. Further, it is observed that the starting point of current draw I_2, I_3 and the end point of I_3 are not captured correctly. The result of such errors could cumulate and lead to incorrect prediction of battery lifetime. Although such errors can be reduced by decreasing the sampling interval Δt, the resulting increase of sampling events will significantly reduce the speed of ns-3 simulations.

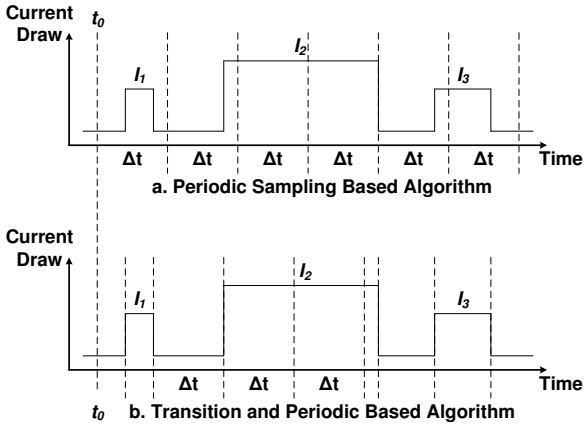


Figure 3: Current draw sampling for update of energy source. (a) shows the sampling scheme used in [7]. (b) shows the sampling scheme used in our work.

```
UpdateEnergySource () {
   Cancel any previously scheduled
      UpdateEnergySource function calls.
   L = CalculateTotalCurrent()
   α_est = discharge (L, present_time)
   if (α_est ≥ α) {
      lifetime = present_time
      battery_level = 0
      NotifyEnergyDrained ()
   }
   else {
      battery_level = 1 − (α_est/α)
      Schedule next call of
         UpdateEnergySource after time Δt.
   }
}
```

Δt is the update interval specified by users.

Figure 4: Algorithm for energy update using R-V battery model, adopted and modified from [7].

In order to overcome this issue, we modified the runtime algorithm by taking a transition based approach in combination with periodic sampling of current draw. The modified algorithm always records transitions in the current draw by calling the *UpdateEnergySource* function. When no transitions occur, the modified algorithm will sample the current draw and update the remaining energy periodically. Figure 3b shows the modified algorithm current draw diagram. It can be observed that each transition of current draw value is captured, and the starting and ending point of current draw values I_1, I_2 and I_3 are all correctly recorded. Fur-

thermore, the user can increase the sampling interval Δt to larger values such that the overall number of calls to *UpdateEnergySource* is reduced, without affecting the accuracy of lifetime estimation. Details of the *UpdateEnergySource* function for the *RvBatteryModel* class are shown in Figure 4.

4.3 Device Energy Model Class

The *DeviceEnergyModel* class implements the energy consumption models of various devices. Each *DeviceEnergyModel* corresponds to a device. For example, we define the *WifiRadioEnergyModel* to represent the energy consumption due to the *WifiPhy* interface on a node, where *WifiPhy* is a class defined in ns-3 to represent the physical layer of the WiFi standard.

As outlined in Section 4.2 the *DeviceEnergyModel* objects of a node are maintained as a list called *DeviceEnergyModelContainer* within the *EnergySource* object. This design is motivated by the the dependency of *DeviceEnergyModel* on *EnergySource*, since device energy consumption cannot occur at a node without an energy source. The *DeviceEnergyModel* is defined as an abstract base class and its child classes are characterized by the actual devices they are modeling.

Devices such as a WiFi radio have several operating states with different energy consumption. Therefore, the *DeviceEnergyModel* is designed to be state based, with a current draw value associated with each of the states. Using current draw to represent each state allows us to calculate the total current at an *EnergySource*, which is required for nonlinear battery models such as the *RvBatteryModel*.

Another important feature included in *DeviceEnergyModel* is to allow the user to specify the behavior of the device when node energy is completely drained. This feature is implemented using a callback defined in child classes of *DeviceEnergyModel*. When energy is completely drained at the *EnergySource*, this callback will be invoked automatically. For example, if the user wishes to investigate the network lifetime, he/she can set the callback to terminate the simulation. Similarly, for network connectivity studies, the callback can be set to disable the WiFi radio of the node, removing the node from the rest of the network. Such design allows greater degree of freedom for the user to conduct simulations for various types of network studies. Further, this callback is assigned by the top level simulation script, allowing the same set of underlying models to be used in different simulation scenarios. By default, a warning message showing energy is completely drained is shown if this callback is not explicitly set by the user.

We define the *DeviceEnergyModel* API as follows:

- *SetEnergySource*: Specifies the *EnergySource* object from which the device model consumes energy.
- *ChangeState*: Notifies the *DeviceEnergyModel* objects of changes in the state of the corresponding devices.
- *GetCurrentA*: Returns the current draw (in *Amperes*) of the device at its present state.
- *HandleEnergyDepletion*: This function is invoked by *EnergySource* when energy is completely depleted. It defines the behavior of the device at the point of energy depletion.
- *GetTotalEnergyConsumption*: Returns the total energy consumed by the device. If called during simulation, it will return the total energy consumed by the device till the instant the function is called.

To introduce a new device model into the framework using this set of APIs, one needs to specify: a) the operating states of the device and b) the current draw figures associated with those operating states. We note that the design of *DeviceEnergyModel* can also be used for devices that do not have finite number of states. For example, in an electric vehicle, the current draw of the motor is determined by its speed. Since the speed of the vehicle can take continuous values within a certain range, it is infeasible to define a set of discrete states of operation. However, the *DeviceEnergyModel* API can still be used by converting the speed value into current draw. This approach has been used in the *GliderEnergyModel* from the *ns-3 UAN Framework* [12].

A child class *WifiRadioEnergyModel*, is derived from the *DeviceEnergyModel* base class to represent energy consumption of a WiFi radio.

4.3.1 WiFi Radio Energy Model

WifiRadioEnergyModel stores the current draw values for each WiFi radio operation state defined in the ns-3 *WifiPhy* class: *IDLE*, *CCA_BUSY*, *RX*, *TX* and *SWITCHING*. By default, *CCA_BUSY* and *SWITCHING* have the same current draw as the *IDLE* state. A similar state definition is also used in [13, 6, 4]. Users can specify these values for a certain WiFi radio based on its data sheet.

To help integrate the *WifiRadioEnergyModel* with existing ns-3 WiFi classes, we implement a listener based on the *WifiPhyListener* class in ns-3. Whenever the WiFi radio changes its state, the listener notifies the *WifiRadioEnergyModel* of the new state of the WiFi radio.

5. VALIDATION AND RESULTS

In this section, validation of the proposed energy framework is presented, followed by a network simulation example that illustrates the usage of the framework. In this simulation example, the network lifetime of a wireless network running the Optimized Link State Routing (OLSR) protocol is investigated. Finally, we illustrate how the proposed framework can be used to implement an energy-aware OLSR protocol that improves network lifetime.

5.1 Framework Validation

The implementation of the framework has been checked via unit tests. The *RvBatteryModel* implementation was validated by comparing the results with those in [11] for the load profiles used therein. Note that the battery lifetime was calculated off-line in [11] given the entire load profile, whereas our implementation calculates the battery lifetime at runtime. As shown in Table 1, simulation results of our *RvBatteryModel* match those in [11].

The *WifiRadioEnergyModel* and *BasicEnergySource* implementations are validated by comparing simulation results with theoretical radio energy consumption calculations. The *WifiRadioEnergyModel* can be further validated by conducting experiments with commonly available WiFi devices such as a laptop computer. A method for measuring WiFi device energy consumption on a laptop computer is proposed in [4]. This will be discussed in our future work.

Load Profile (mA)	Battery 1 Lifetime (min.)			Battery 2 Lifetime (min.)		
	[11]	Our	diff.	[11]	Our	diff.
C1	36.2	36.2	0	55.0	55.0	0
C2	55.8	55.7	0.1	73.9	73.9	0
C3	71.8	71.7	0.1	88.8	88.7	0.1
C4	124.9	124.3	0.6	137.8	137.8	0
C5	176.7	176.0	0.7	185.8	185.7	0.1
C6	41.0	41.0	0	58.9	58.9	0
C7	30.8	30.9	0.1	51.1	51.1	0
C8	37.4	37.4	0	55.0	55.0	0
C9	35.2	35.2	0	55.0	55.0	0
C10	132.6	132.1	0.5	144.3	144.3	0
C11	107.4	107.3	0.1	144.3	144.3	0
C12	155.4	154.9	0.5	169.3	169.2	0.1
C13	131.7	131.0	0.7	144.3	144.3	0
C14	126.3	126.3	0	141.5	141.6	0.1
C15	209.2	208.6	0.6	211.4	211.4	0
C16	200.7	200.0	0.7	211.4	211.4	0
C17	251.2	250.4	0.8	261.4	261.4	0
C18	204.6	203.1	1.5	211.4	210.3	1.1
C19	208.7	207.1	1.6	216.4	215.2	0.8
C20	33.2	33.2	0	55.3	55.2	0.1
C21	55.9	55.9	0	79.6	79.5	0.1
C22	94.5	94.5	0	112.2	110.9	1.3

Table 1: Battery lifetime comparison between results in [11] and our implementation of the R-V battery model for the same load profiles.

5.2 Simulation Example

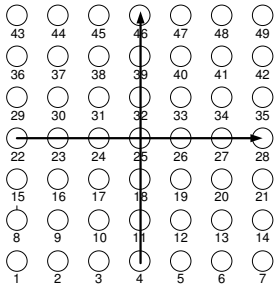
In this section, the proposed framework is used to investigate the lifetime of a wireless network running the OLSR protocol. We define the lifetime of a network as the time of the first node failure in the network. Node failures are assumed to occur only when energy of a node is completely drained.

5.2.1 Setup

The network topology used in the simulation consists of 49 nodes, arranged in a 7 × 7 grid, as shown in Figure 5. The radio range and the inter-node distance of the grid are both set to 650 m under Friis free space propagation model, which ensures that only nodes that are vertically or horizontally adjacent can communicate directly with each other. Nodes 4 and 22 are set as UDP sources, sending CBR traffic at 16 kbps to destination nodes 46 and 28 respectively. The routing protocol used is OLSR, with the HELLO packet [5] interval set as 2 s.

A *BasicEnergySource* object is installed with initial energy of 100 *Joules* on each node, and the node's remaining energy is monitored throughout the simulation. The node is declared as failed once the remaining energy reaches zero. A *WifiRadioEnergyModel* object is also installed on each node to capture the WiFi radio energy consumption. Based on the power consumption for typical WiFi radios [6], the transmit current, receive current and idle current are set to 85.7 mA, 52.8 mA and 18.8 mA respectively. In this simulation, energy consumption due to other devices on the node is omitted.

We present results for two versions of this simulation,



Grid Spacing	650m
Data Rate	16 kbps
Traffic Contract	CBR
Transport Protocol	UDP
Routing Protocol	OLSR
Initial Energy	100 Joules
Transmit Current	85.7 mA
Receive Current	52.8 mA
Idle Current	18.8 mA

Simulation Parameters

Figure 5: Network topology and simulation parameters used for the OLSR simulation. UDP traffic flows from node 4 to node 46, and from node 22 to node 28.

based on the value of the willingness parameter in OLSR. According to OLSR standard [5], the value of a node's willingness parameter is an integer between 0 and 7, and indicates its willingness to carry traffic on behalf of other nodes. A willingness of 7 implies that a node will always relay other nodes' packets, while a willingness of 0 means that it never participates in data traffic relay.

In the first version, the willingness parameter of the OLSR protocol is set to the default value of 3 for all the nodes, and is constant throughout the simulation. In the second version, the willingness of a node based on its available residual energy using the following equation:

$$W_n = floor\left(\frac{\frac{RE_n}{IE_n} \times 100}{15}\right) \quad (2)$$

where W_n is the willingness of node n, RE_n and IE_n denote the remaining energy and initial energy at the energy source of node with index n, respectively. Equation 2 bounds W_n between 0 and 7, as required by OLSR standard. W_n decreases from 7 to 0 as the energy depletes.

5.2.2 Results: Constant Willingness

In this version, the traffic flows follow the paths shown in Figure 5. This leads to high energy consumption at node 25, since it relays packets from both streams. Even when the energy at node 25 is much lower than its neighbors, the OLSR protocol continues to relay packets through it, and the energy depletes further, until finally, node 25 fails and the network lifetime is reached. The lifetime of the network in this case was observed to be 1164.8 s. The energy consumption at the different nodes is shown in Figure 6 in terms of remaining battery level. We note that nodes 17, 19, 31 and 33, although immediately adjacent to the traffic paths, do not participate in relaying data, and hence have significant energy remaining.

5.2.3 Results: Energy-aware Willingness

In this version, traffic is expected to be re-routed as the energy of node 25 reduces, thus increasing its time to failure, and hence the overall network lifetime. First, we run this simulation for 1164.8 s (which is the lifetime of the constant willingness simulation) and observe the remaining energy of all the nodes. The results, as seen in Figure 6, show that node 25 still has some energy remaining in its battery. As expected, its willingness parameter is 0, and traffic is routed around it. Further, we measured the network lifetime in this

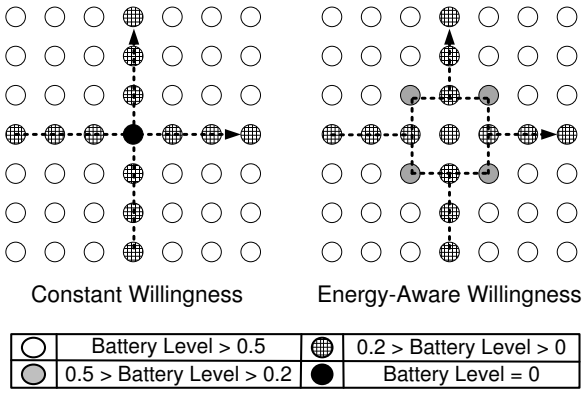


Figure 6: Traffic routes and energy consumption for the constant willingness and energy-aware willingness versions of OLSR. The energy-aware version partially redistributes energy consumption, which improves the lifetime of node 25, and hence, of the network.

version by running a longer simulation (2000 s) and found that node 25 is again the first one to fail, at 1232.8 s. Ideally, we would expect greater improvements in lifetime, but the energy at node 25 keeps depleting as a result of idle listening of packets being relayed by its 1-hop neighbors. This is due to the broadcast nature of wireless communication and the moderate size of network topology. Previous study [6] has also shown that idle listening contributes the majority of the energy consumption in WiFi (IEEE 802.11) networks.

6. EXTENSION AND FUTURE WORK

In this section, we discuss possible extensions to the proposed energy framework.

6.1 Computation Energy Consumption

Currently ns-3 focuses only on simulating network related events such as sending and receiving of packets. Computation tasks on a node are not considered as simulation events and there is no notion of computation time or delay. However, it is desirable to include the cost computation into the proposed energy framework for conducting more realistic energy simulation of a network.

The design of the *DeviceEnergyModel* class allows the user to define child classes for modeling computation energy consumption of network nodes. A child class of *DeviceEnergyModel* can be created containing states representing computation tasks available on the node. For example, a simple computation energy model for a node could contain an idle state and one or more active states. As shown in [14], the current draw figures of each of these states can be obtained by simulation or experiment. Given the current draw values and the time spent in each state, existing *DeviceEnergyModel* interfaces can be used to incorporate computation energy consumption into ns-3 simulations.

6.2 Energy Scavenging

Recently, scavenging energy from sources such as solar, thermoelectric and human inputs has been proposed for powering mobile devices [9]. In applications such as wearable computing and sensor networks, scavenged energy may become the primary energy source for wireless devices. Further, energy scavenging techniques can be used in combination with traditional energy sources such as batteries. Including energy scavenging techniques in ns-3 will involve constructing models for their behavior and integrating them with the proposed energy framework. Also, new interfaces may need to be added to the *EnergySource* API described earlier in Section 4.2. Once energy scavenging is integrated with the proposed energy framework, researchers will be able to construct simulations for nodes that are powered solely by energy scavenging techniques or by a combination of traditional energy sources and energy scavenging.

7. CONCLUSION

In this paper, we presented an integrated, modular and easily extensible energy framework for ns-3. The proposed framework can be used to model linear and non-linear energy sources as well as the energy consumption of wireless devices in ns-3. Through a simulation example, we showed how our framework enables users to not only measure energy consumption in wireless networks but also use this information to implement energy-aware protocols that improve network performance.

8. ACKNOWLEDGEMENT

This work was supported by ARO MURI Grant W911NF-07-1-0287.

We would like to thank Dr. Tom Henderson for his help during the development of the ns-3 Energy Framework. We would also like to thank Mathieu Lacage, Nicola Baldo and Andrea Sacco for their efforts in reviewing the ns-3 Energy Framework code.

9. REFERENCES

[1] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/index.html.

[2] The ns-3 network simulator. http://www.nsnam.org/.

[3] Opnet modeler. Opnet Technologies Inc., http://www.opnet.com.

[4] Atheros Communications. Power Consumption and Energy Efficiency Comparisons of WLAN products. *White paper*, 2003.

[5] T. Clausen and P. Jacquet. IETF RFC-3626: Optimized Link State Routing Protocol OLSR. *The Internet Society http://www. ietf. org/rfc/rfc3626.txt*, 2003.

[6] M. Ergen and P. Varaiya. Decomposition of Energy Consumption in IEEE 802.11. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 403–408, 2007.

[7] M. Handy and D. Timmermann. Simulation of mobile wireless networks with accurate modelling of non-linear battery effects. In *Proc. of Applied simulation and Modeling (ASM)*, 2003.

[8] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi. Battery-driven system design: a new frontier in low power design. In *Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design. Proceedings.*, pages 261–267.

[9] J. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *Pervasive Computing, IEEE*, 4(1):18–27, 2005.

[10] D. Rakhmatov and S. Vrudhula. An analytical high-level battery model for use in energy management of portable electronic systems. In *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, pages 488–493.

[11] D. Rakhmatov, S. Vrudhula, and D. Wallach. Battery lifetime prediction for energy-aware computing. In *Low Power Electronics and Design, 2002. ISLPED '02. Proceedings of the 2002 International Symposium on*, pages 154–159, 2002.

[12] A. Sacco and L. Tracy. GSOC2010UANFramework. 2010. http://www.nsnam.org/wiki/index.php?title=GSOC2010UANFramework.

[13] M. Spohn, S. Sausen, F. Salvadori, and M. Campos. Simulation of blind flooding over wireless sensor networks based on a realistic battery model. In *Networking, 2008. ICN 2008. Seventh International Conference on*, pages 545–550, 2008.

[14] T. Tan, A. Raghunathan, and N. Jha. Emsim: an energy simulation framework for an embedded operating system. In *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, volume 2, pages II–464–II–467 vol.2.