Energy Framework: An Extensible Framework for Simulating Battery Consumption in Wireless Networks

Laura Marie Feeney
Communication Networks and Systems Lab
Swedish Institute of Computer Science
lmfeeney@sics.se

Daniel Willkomm
Telecommunication Networks Group
Technische Universität Berlin, Germany
willkomm@tkn.tu-berlin.de

ABSTRACT

Energy Framework is an extensible OMNeT++ framework for modeling battery consumption in wireless networks. It is designed so there is a clear separation between modeling of battery state and energy consuming operations, enabling more sophisticated modeling of system behavior. The Energy Framework is largely implemented in "pure" OMNeT++; it has been successfully incorporated into the Mobility Framework and MiXiM.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: Miscellaneous

General Terms

Measurement

Keywords

OMNeT++, battery model, wireless network

1. INTRODUCTION

This paper presents the Energy Framework, an extensible OMNeT++ [9] framework for modeling battery consumption in wireless networks, particularly sensor networks. The Energy Framework is highly modular, separating modeling of the internal state of the battery from the modeling of operations that consume energy. This structure enables functionality that is not easily supported in other models, including multiple sources of energy consumption, explicit battery state-of-charge estimation, and a variety of abstractions for energy consuming operations.

The Energy Framework uses only generic OMNeT++ features, with the exception of handling host failure due to battery depletion, which is framework specific. Correctly handling changes in host state is non-trivial because the notification must be delivered to all modules, not just those that interact directly with the battery, to ensure that statistics being collected by modules are consistent. With this in mind, the Energy Framework has been incorporated into the Mobility Framework [2] and into MiXiM [4], where this issue has been carefully addressed.

More generally, the Energy Framework is intended to enable the development of more sophisticated models of both internal battery state and energy consuming devices and to better represent the battery as a complex and dynamic element of the system. The Energy Framework is highly extensible and portable; this is facilitated by an extensive collection of sample devices and validation tests.

OMNeT++ 2010 March 15–19, Torremolinos, Malaga, Spain.

2. BACKGROUND AND RELATED WORK

Managing battery consumption is essential for the usefulness of many kinds of wireless networks, where the opportunity to recharge devices may be limited and where devices must act together to implement some distributed functionality. Sensor networks present particularly interesting challenges, because of the variety of hardware used and because of their extremely demanding lifetime constraints.

Sensor networks are modeled in many general purpose network simulation platforms, such as ns-2, ns-3 [5] and OMNeT++, as well as in hardware/OS-specific platforms such as PowerTossim [6, 8]. Existing support for battery consumption modeling in OMNeT++ includes [1] and [3]: The model in [1] is similar to Energy Framework in supporting multiple energy consuming devices, but has only limited support for handling battery depletion. The model in [3] is tightly integrated with the radio model and only considers the energy consumption of the transceiver.

For both network- and hardware-oriented simulation environments, the underlying battery consumption model is essentially based on "operation counting": estimating the number of (or amount of time spent on) various operations. Combined with detailed measurements of these operations on the hardware platform in question, it is possible to compute the power consumed by each operation and, by extension, of running an application. The limitation of this approach is that it simply assigns a fixed cost to each operation and subtracts that cost from a counter representing the battery capacity, rather than modeling the battery's internal state. (PowerTossimZ [8] is a partial exception, as the total energy consumption is computed offline in post-processing using a table-driven stochastic model.)

The goal of the Energy Framework, in contrast, is to model the battery as an active element in the simulation, with the ability to reflect the time-varying and non-linear behavior that it would have in reality. In the first phase of this work, we have focused on developing sound mechanisms

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8725
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8725

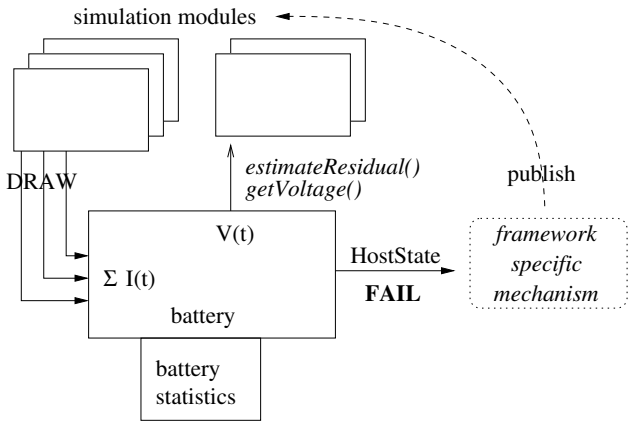

Figure 1: Energy Framework: Architecture

for handling host failure due to battery depletion.

3. DESIGN AND IMPLEMENTATION

In this section, we describe the architecture of the Energy Framework and the sample battery model implemented in the *SimpleBattery* module.

3.1 Architecture

The per-host architecture of the Energy Framework is shown in Figure 1. It is centered on the battery module, which receives DRAW messages (or function calls) from one or more energy consuming devices, applies a battery consumption model to compute the internal state of the battery, and eventually announces battery depletion. Note that "device" here refers to any simulation module that models an energy consuming operation (or even an energy producing one, as in a solar panel) and informs the battery of the associated current draw; it does not refer to a specific base class.

The battery module takes as input a sequence of DRAW messages, defining a step function for the current draw of each device. The sum of these functions is also a step function, defining the total current draw. The battery consumption model implemented by the battery module uses this (and possibly other data such as ambient temperature) to model the internal state of the battery and to determine when the battery becomes depleted.

The battery module is also responsible for maintaining two public interfaces, *estimateResidual()* and *getVoltage()*. The *estimateResidual()* interface may be used by any module that simulates a "battery-aware" mechanism, such as load balancing. The return value represents a host's own real time estimate of its battery's residual capacity. It may therefore differ in precision and accuracy from the simulated battery internal state, reflecting the limitations of battery state-of-charge estimation. The *getVoltage()* interface may be used by any module that models the effect of battery voltage on hardware behavior; it is meaningful only if the underlying battery model supports voltage modeling.

The Energy Framework is largely framework independent. The exception is the mechanism by which the battery module notifies simulation modules of host failure due to battery depletion. This functionality is needed to ensure not only that a failed host no longer communicates on the network, but also that other internal logging and statistics are correct. Host failure notification may therefore need to be handled by modules that are not themselves battery aware: A "zombie" application module that continues to generate messages after the host has failed may collect invalid statistics (e.g. total application packets), even if the NIC module is correctly disabled. This issue is discussed below.

The battery statistics module handles logging of internal battery state. The canonical mechanism is the OMNeT++ vector, but can also be a log file or GUI. It is kept separate from the battery module because output formatting is the functionality most likely to be changed by non-expert users. Moreover, protecting access to the battery's internal state helps enforce that other simulation modules use the battery modules's *estimateResidual()* interface.

The Energy Framework is designed to provide flexibility and extensibility. Providing a well-defined interface to a battery module separates the modeling of the internal state of the battery from the modeling of operations that consume energy. This approach has several advantages: It is straightforward to represent a host with multiple independent devices that consume energy, e.g. a transceiver, a sensor, and an actuator. There is also explicit support for real time state-of-charge estimation. Furthermore, each device can be modeled differently: its operation can be modeled in detail or it can be represented by an abstract device, such as the DutyCycle device, that simply generates an appropriate pattern of current draw.

This approach also facilitates code re-use by isolating expert knowledge in the appropriate module. We expect there will be a very small number of battery modules, encapsulating models of substantial complexity and domain expertise. At the same time, the implementation of the much larger number of device modules is simplified, because modeling of state changes and operations that consume energy is embedded in the device module, not the battery module.

Obviously, when building a simulation, the user must take care to ensure comparable fidelity between modeling of energy consuming operations and modeling of battery state. For example, it makes little sense to combine a simplified model of the operation of a network interface with a detailed electro-chemical battery model, and vice verse.

3.2 SimpleBattery

The *SimpleBattery* module provides a simple, linear model of battery consumption similar to that found in most of the systems described above. It provides a fairly coarse estimate of battery consumption, with little computational overhead.

SimpleBattery maintains a table with the amount of current I_d being drawn by each device d and updates its residual capacity E according to:

$$E = E - \sum_d I_d \times \Delta t \times V_{\text{nominal}}.$$

where Δt is the time since the last update and is bounded by the resolution of the battery model, giving an upper limit on the delay in detecting battery depletion. In this simple model, the voltage is assumed to be constant at the nominal value V_{nominal} until battery depletion.

Whenever the amount of current being drawn by a device changes, the device module sends a DRAW-CURRENT message to the battery. The residual capacity is updated using the formula above and the new CURRENT value I_d is then written into the table.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8725
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8725

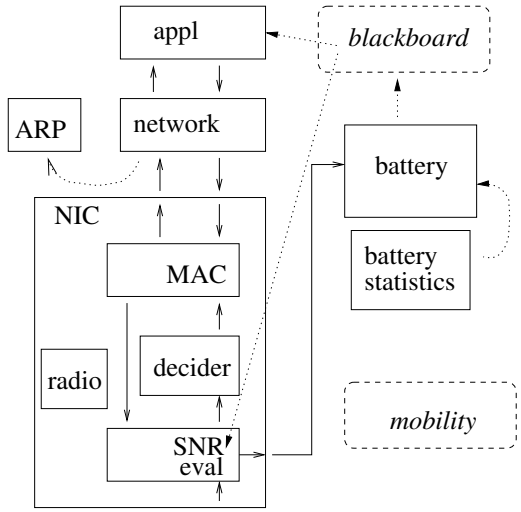

Figure 2: Mobility framework: IEEE 802.11 NIC

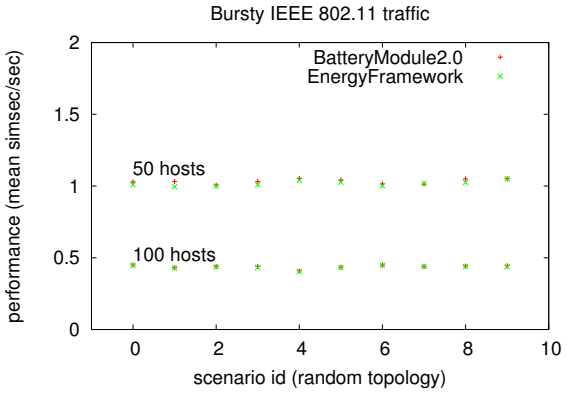

Figure 3: Comparing the Energy Framework with an integrated solution, BatteryModule2.0. The performance difference is negligible; the data points generally overlap.

SimpleBattery further extends the battery module by providing an additional message, DRAW-ENERGY, which causes a discrete subtraction from the battery capacity. Although such instantaneous energy consumption does not happen in reality, it is an efficient way to account for the energy consumption of operations that are only modeled abstractly in the simulation or that have a complex current draw over a short time. Examples include state transitions such as waking up the radio transceiver or the operation of sensor/actuator hardware.

SimpleBattery is also meant as a sample implementation of an Energy Framework battery module, making integration of alternative battery models relatively straightforward.

4. MOBILITY FRAMEWORK

In the next two sections, we describe the integration of the Energy Framework into two widely-used wireless network simulation frameworks: Mobility Framework and MiXiM.

4.1 Implementation

The Energy Framework was originally implemented and tested in the Mobility Framework [2]. Two modules, CSMA and IEEE 802.11 NIC, have been modified to send DRAW messages to the *SimpleBattery*. The integration of the two frameworks is shown in Figure 2.

The *SimpleBattery* module uses the Mobility Framework's publish-subscribe BlackBoard to send HostState notifications. When the residual capacity becomes negative, a HostState notification is published. The NIC's SnrEval module and the Application Layer module both subscribe to HostState notifications. The former deletes all frames arriving at the NIC. The latter stops generating new application frames.

It is not currently possible to enforce that independently developed Mobility Framework modules subscribe and respond properly to a change in Host State. This problem is fully addressed in the MiXiM port.

4.2 Performance

The Energy Framework is intended to be highly extensible and uses OMNeT++ message passing to inform the battery module of changes in the current being drawn. We have explored some of the advantages of this approach, but it is also important to consider the overhead compared to more integrated approaches. We therefore compare the performance of the Energy Framework to that of BatteryModule2.0 [3], another contributed module for the Mobility Framework.

BatteryModule2.0 uses the Mobility Framework's BlackBoard to snoop on radio state changes and decrements the residual battery capacity according to a linear model like that used in *SimpleBattery*. This integrated strategy means that it is not possible to include other sources of energy consumption, nor does BatteryModule2.0 handle host failure.

BatteryModule2.0 was easily reimplemented in the Energy Framework by creating a new Device module that also uses the BlackBoard to detect radio state changes, but sends *Draw* messages to the Energy Framework's *SimpleBattery* module. A new *BatteryStatistics* module was also implemented to collect data from the *SimpleBattery* and write it to per-host log files, as in BatteryModule2.0. It was confirmed that the two variants obtained the same results.

The performance comparison used the Mobility Framework's bursty traffic application module and IEEE 802.11 NIC, with no mobility. The performance metric is the number of simulated seconds per second (simsec/sec), as sampled by the OMNeT++ engine every 50000 events, taking the harmonic mean rate over the simulation lifetime. The first and last samples were discarded to avoid initialization and finalization effects.

Figure 3 shows the results. Ten random topologies were generated for networks of 50 and 100 simulated hosts. For the smaller network, a BatteryModule2.0 simulation runs, on average, 1.5% simsec/sec faster than an equivalent simulation in the Energy Framework. As the number of hosts (node density) doubles, the difference is just under 1%. Measured CPU time gives similar results. The data suggest that the Energy Framework is reasonably performant compared to the more integrated solution.

5. MIXIM

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8725
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8725

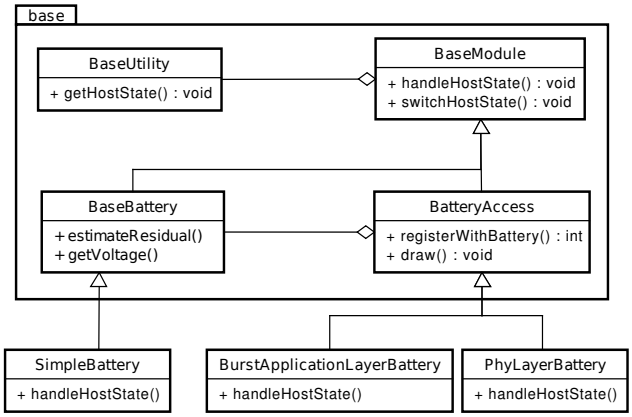

Figure 4: MiXiM: Host state and energy architecture

Energy Framework has been ported to MiXiM [4], where it was released as part of MiXiM 1.1. Following the general philosophy of MiXiM, the Energy Framework was separated into core functionality (*BaseBattery* and *BatteryAccess* modules) and functionality that is (re)implemented by users (*SimpleBattery* and *BatteryStats* modules; use of *draw* method). The integration into MiXiM is shown in Figure 4.

BaseBattery takes care of the generic functionality described in Section 3.1. *BatteryAccess* provides access to the *draw()* function implemented in *BaseBattery*. In MiXiM, we decided to use function calls to inform the battery of current draw, rather than using OMNeT++ messages. The main reasons for this design decision are simplicity and scalability. Because the Energy Framework is deeply integrated into MiXiM, there is no structural advantage to using messages. Moreover, using function calls requires even less overhead than the BlackBoard mechanism used in the BatteryModule2.0 (Section 4.2). Sub-classing from the base classes makes it easy to add energy consumption to MiXiM modules or to implement more sophisticated energy modules.

As in the Mobility Framework, host failure is handled via the BlackBoard (*BaseUtility* in MiXiM) using a *HostState* parameter. MiXiM, however, addresses the problem of host wide notification better, allowing for excellent integration with the Energy Framework and making it possible to safely use battery aware modules together with modules that do not necessarily model battery consumption, but are affected by a change in the Host State.

MiXiM guides the user by ensuring that each MiXiM module (*BaseModule*) is automatically subscribed to *HostState*. The default action for a change of *HostState* to something other than ACTIVE is to throw an error, telling the user that the module is not capable of handling *HostState*. The user must either overwrite the *handleHostState* function to provide module-specific functionality or explicitly set the *notAffectedByHostState* parameter for that module. This mechanism is also a good basis for implementing other uses of host state notification, such as low-power sleep modes.

6. CONCLUSION

We have described the Energy Framework for OMNeT++, an extensible framework for simulating battery consumption, and its instantiation in two wireless network simulation frameworks. Our approach of separating modeling of battery internal state and energy consuming operations considerably simplifies modeling of device hardware and makes it easy to expand the range of devices that model their energy consumption. Improved handling of host state notification, particularly in MiXiM, makes it practical for simulations to include modules that are not explicitly battery aware.

Our primary goal is to increase the sophistication of the battery consumption and state-of-charge estimation models, such as described in [7]. The development of real-time battery models that are computationally feasible for network-scale simulation (i.e. many hosts/batteries over a long period) remains a challenging open problem. The Energy Framework's extensibility facilitates this investigation by making it easy to integrate, compare, and evaluate models.

The software referred to in this paper is available via `www.omnetpp.org`. We encourage the community to experiment with it and we particularly invite collaboration in the development and refinement of battery and device models.

Acknowledgments

Parts of this study were carried out within the VINN Excellence Center WISENET, partially funded by VINNOVA, the Swedish Governmental Agency for Innovation Systems. The authors would also like to thank Karl Wessel for his contributions to porting the Energy Framework to MiXiM.

7. REFERENCES

[1] I. Dietrich, F. Chen, R. German, and F. Dressler. Modeling energy consumption of wireless communications in OMNeT++. GI/ITG KuVS Fachgespräch Systemsoftware und Energiebewusste Systeme, Oct. 2007.

[2] W. Drytkiewicz, S. Sroka, V. Handziski, A. Koepke, and H. Karl. A mobility framework for OMNeT++. In *3rd Int'l OMNeT++ Workshop*, Jan. 2003.

[3] A. Förster. BatteryModule 2.0. `http://www.inf.unisi.ch/phd/foerster/downloads.html`.

[4] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K. Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in OMNeT++: the MiXiM vision. In *Simutools '08: Proceedings of the 1st Int'l Conf on Simulation tools and techniques & Workshops*, 2008.

[5] ns-3. `http://www.nsnam.org`.

[6] E. Perla, A. O. Catháin, R. S. Carbajo, M. Huggard, and C. Mc Goldrick. PowerTOSSIM z: Realistic energy modelling for wireless sensor network environments. In *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks (PM2HW2N '08)*, 2008.

[7] R. Rao, S. Vrudhula, and D. N. Rakhmatov. Battery modeling for energy-aware system design. *Computer*, 36(12), 2003.

[8] V. Shnayder, M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys '04: Proceedings of the 2nd Int'l Conf on Embedded Networked Sensor Systems*, 2004.

[9] A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, June 2001.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8725
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8725