PowerTOSSIM z: Realistic Energy Modelling for Wireless Sensor Network Environments

Enrico Perla
School of Computer Science and Statistics
Trinity College Dublin, Ireland
perlae@cs.tcd.ie

Art Ó Catháin
School of Computer Science and Statistics
Trinity College Dublin, Ireland
ocathaia@cs.tcd.ie

Ricardo Simon Carbajo
School of Computer Science and Statistics
Trinity College Dublin, Ireland
carbajor@cs.tcd.ie

Meriel Huggard
School of Computer Science and Statistics
Trinity College Dublin, Ireland
Meriel.Huggard@cs.tcd.ie

Ciarán Mc Goldrick
School of Computer Science and Statistics
Trinity College Dublin, Ireland
Ciaran.McGoldrick@cs.tcd.ie

ABSTRACT

Energy continues to be the key constraint in wireless sensor networks. We review existing methods for estimating software power consumption and battery modelling, as applied to embedded systems such as Wireless Sensor Networks. We consider current developments in hardware and software technology, in particular the availability of high-fidelity simulators. Once such simulator, TOSSIM for TinyOS 1.x, models power consumption via a plugin, PowerTOSSIM. We complete the port of PowerTOSSIM to TinyOS 2.0 for the latest model of sensor node, the MICAz. Finally, we extend the simulator to model non-linear battery effects.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete Event*

General Terms

Design

Keywords

TinyOS 2, TOSSIM, PowerTOSSIM, energy model, MICAz, wireless sensor networks, non-linear battery model

1. INTRODUCTION

Applications that make use of wireless sensor networks have multiplied in recent years. Power consumption is perhaps the key issue in wireless sensor network design, and much current hardware and software research is devoted to increasing the lifetime of sensor motes. Hardware advances have seen the development of a new family of motes, the MICAz [11] model, which has better performance and lower radio battery consumption than its predecessor, the MICA2 [10].

From the software point of view, an event-based model that avoids busy-waiting is at the heart of the design of TinyOS [24]; the first and most popular operating system in this environment. Thus the design of this operating system has been directly influenced by power considerations. Another issue that affects power consumption is the positioning of the motes and the environment in which they act. For example, the impact of network topology on routing algorithm performance may show that a specific routing algorithm overstresses the battery in some configurations while performing well in others. Testing the relationship between the individual mote's battery discharging rate and its effective environment is therefore of fundamental importance.

To perform these tests on physical hardware poses a range of problems including financial investment, the reproducibility of the environment, and the potential impact of hardware failure. Simulators are a cheap, reliable and effective alternative. To this end, TinyOS has a simulation environment TOSSIM [26]. The previous version of TOSSIM included PowerTOSSIM [16], a dedicated plugin which models power consumption. However this plugin has not been fully ported to the new TOSSIM v2. In particular, it does not simulate MICAz motes.

The format of this paper is as follows: A general overview of wireless sensor networks is given, including a brief history of their origins. Next the the current state of the art in sensor node hardware is outlined and TinyOS, TOSSIM and PowerTOSSIM are described. Our modifications of TOSSIM for TinyOS 2.0 are then detailed. Finally, a brief overview of current research into non-linear battery effects in embedded devices and our integration of this research into PowerTOSSIM z is provided.

2. WIRELESS SENSOR NETWORKS

Research into wireless sensor networks began in the 1980s under the Distributed Sensor Networks program of DARPA (Defense Advanced Research Project Agency) [5]. Such a

PM²HW²N'08, October 31, 2008, Vancouver, BC, Canada.

Figure 1: MICAz mote

network consists of sensor nodes and optionally, a base station. A sensor node [9] comprises a microprocessor, data storage, sensors, analog-to-digital converters, a data transceiver (radio), and an energy source. Each node is small, lightweight and portable.

2.1 Energy Limitations

The small size and long operational life requirements of the sensor nodes lead to energy limitations. Radio communication is the primary source of energy consumption. To save energy, and to minimize power consumption, devices should be turned off or set to sleep mode when possible.

Power management is essential to the future success of wireless sensor networks. A key principle for low power consumption and better power management is to use sleep mode here possible. The current drawn by a device in sleep mode can be reduced by:

- isolating and turning off the individual circuits
- waking up when required
- quickly starting processing and active mode
- minimizing the work to be done
- returning to sleep after the work is done.

2.2 Hardware Devices

The MICA family of WSN motes was introduced in 2001. The original MICA motes were designed by the Department of Electrical Engineering and Computer Sciences at UC Berkeley, California. Manufacturing and marketing was handled by Crossbow, a private company also based in California. The followups, MICADot2 and MICA2, first appeared in 2002. They featured increased power and functionality without sacrificing power consumption. The full-sized motes use 2 AA batteries which can provide power for upwards of one year depending on the application. The MICA range support sensor and data acquisition boards; the boards are connected to the motes using a 51-pin expansion connector.

The latest addition to the MICA range is the MICAz, featuring an improved radio module and hardware AES encryption. It uses the ATmega 128L MCU (MicroController Unit). Its Chipcon CC2420 wideband radio supports the new 802.15.4 / ZigBee protocols. The data rate is increased to 250Kbps which is an improvement on the 19Kbps data rate of the MICA2. The power consumption of the MICA2 motes have been accurately measured [19] [2].

2.3 TinyOS

TinyOS is the most commonly used operating systems on WSN motes. From the architectural point of view, it is an event-driven, component-based framework: independent components are linked together to build the final application to load on the mote and higher level components communicate with lower level ones by issuing commands and waiting for events to be signaled. Commands and events are decoupled and non-blocking, so that the command returns immediately and the response is signalled afterwards.This behaviour is usually defined as split-phase: the invocation (call) and the completion (signal) of an operation are distinct with two different execution times.

Long term operations, or tasks, form a third abstract computational unit and are scheduled on a FIFO basis. A task can be posted on the *runqueue* by a command, an event or another task (a task can repost itself). This design allows for very long computations to be split into multiple tasks that will post by themselves the following task. The current version of TinyOS is 2.x. This includes some significant improvements over the 1.x, especially in task handling.

2.4 TOSSIM

TOSSIM, the TinyOS Simulator, exploits TinyOS's hierarchical model by replacing lower level hardware components with software emulated ones. TOSSIM maps directly to the TinyOS code: compiling an application for the simulation environment is as simple as compiling it for the real mote (`make micaz sim` vs. `make micaz`). This approach reduces the gap between the simulator and the real environment. By replacing low level components, high fidelity between the simulation environment and reality is achieved.

Good scalability is a natural consequence of the TinyOS design: mote based applications are usually small in size and each internal component has its own private and static frame, thus simplifying simulation overhead needed to keep hundreds of simulated nodes in memory.

In TOSSIM, transitions from an event to another happen instantaneously and so there is no tracking of the execution time. This design model and the lack of simulation code for some low level devices have been the major challenges to be addressed by those wishing to add power estimation primitives to TOSSIM.

So far, two projects have tried to reach this goal: PowerTOSSIM, written for TinyOS/TOSSIM 1.x and for the MICA2 family of motes and PowerTOSSIM 2 [17], a port of the former to work inside TinyOS/TOSSIM 2.x.

3. BATTERY MODELLING

3.1 Nonlinear effects

Modern batteries are analogue, chemical devices. There are several competing chemistries; the most appropriate for a given application depends on the desired tradeoff between such factors as capacity, weight, volume, self-discharge rate, rechargeability, cycle life, safety, etc. Batteries are rated in ampere-hours (Ahr), but this rating is nominal, and in practice the capacity varies according to a number of usage factors. Additionally, the capacity of cells manufactured within

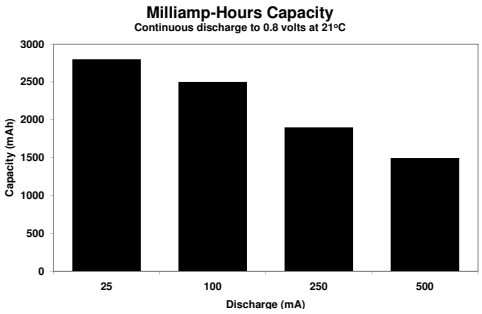

Figure 2: The rate capacity effect: AA battery life at different constant currents. Manufacturer's figures from [7]

the same batch can vary considerably (by up to 20% in some cases).

There are several characteristics of batteries that should be incorporated into an accurate model [18]:

- The voltage of a cell decreases monotonically with time. This relationship is non-linear and depends on the particular chemistry used. Eventually the voltage drops too low for the battery to be usable and the remaining capacity is not accessible.
- Higher discharge rates lead to lower capacity (the Rate Capacity Effect).
- The same total energy load can have a different effect on remaining battery capacity, depending on the shape of the curve of current versus time.
- Batteries recover some of their charge when in the idle state (the Recovery Capacity Effect).

In practical applications, batteries are connected to circuits using a DC-DC converter that ensures the voltage delivered to the circuit is constant. This converter is a source of losses, typically $10 - 20\%$ of the delivered power.

3.2 Modelling approaches

The modelling of batteries can be performed at a high or low level. Circuit-level models, such as SPICE [21], operate at the analogue physical hardware level. In this model continuous-time partial differential equations are solved by numerical analysis. This is a highly accurate method, but a major drawback is that the load model must also be simulated at this level - this is a non-trivial task for modern integrated circuits, requiring intricate knowledge of the processor subsystems. It is also very computationally intensive, and hence does not scale to simulations of a large numbers of nodes.

An alternate discrete-time approach is taken in [13] and extended in [18]. The battery's state is simulated by stochastic modelling of its remaining charge. Broadly speaking, the battery is considered to have a certain number of charge units, and during each time unit the battery's stock is depleted by a quantity conistent with the demand placed on the device. Simultaneously, it has a certain probably of recovering a charge unit. This method is fast and therefore scales well to large numbers of nodes.

4. POWERTOSSIM Z

In this section we describe how hardware power consumption has been incorporated into our implemenation of Power TOSSIM z. Capturing the power consumption of an application running on a given mote necessitates tracking of the behaviour of the mote's low level components, such as the microcontroller and the radio chip. An energy estimator must also take into account the interfaces that TinyOS exports to manage them, and their support within TOSSIM.

4.1 TinyOS interfaces

The TinyOS 2.x power management interfaces are a major improvement over those implemented in TinyOS 1.x. As stated in TEP 112 [22], TinyOS 1.x essentially relied on the application itself to handle the power on/power off states of all the devices. This was accomplished through the use of the `StdControl` interface, which exports a `start` and a `stop` command that any component can call over a given peripheral. This approach simplified the design of PowerTOSSIM 1.x, since most of the calls could then be placed where this interface was implemented, to gain a complete view of the mote's power state.

TinyOS 2.x operates differently in that it divides the devices in two classes:

- the microcontroller, which fundamentally has enough information to independently calculate the power state to use, and
- the peripherals, which have simpler semantics (with the partial exception of the CC2420 Radio Chip, as we will see later on) and two basic power states, on and off.

The relevant power management interfaces and their associated hardware are described in more detail below.Only the devices relevant to our PowerTOSSIM z implementation will be analyzed, with particular emphasis on TOSSIM support and its limitations. This description is at the heart of the design goals and implementation of PowerTOSSIM z, and any future improvements.

4.2 Microcontroller Power Management: The ATM128 MCU

"The Atmega128 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture" [1]. It features six software selectable power-save states (shown in Table 1) which range from the `IDLE` state, which stops the CPU while leaving all the other components active, down to the `POWERDOWN` state, which disables all the components until the next interrupt or hardware reset. TinyOS exports one interface for the handling of the microcontroller, which is called `McuSleep`, which exposes a single asynchronous command, `sleep()`, that is called inside the TinyOS scheduler when the FIFO taskqueue is empty.

The job of the `sleep()` command is to calculate the correct power state in which to put the microcontroller. The task requires analysis of the state of different registers. This is illustrated in the code snippet from the atm128 implementation in Figure 4.2.

Table 1: Atmega128 Power states

Power State	Current
Active	8mA
Idle	4mA
Standby	1mA
ADC Noise Reduction	1mA
Extended Standby	160μA
Power-save	9μA
Power-down	0.3μA

```
< tos/chips/atm128/McuSleep.nc >
[...]
if (TIMSK & ~(1 << OCIE0 | 1 << TOIE0 |
            1 << TOIE1 | 1 << TOIE2) ||
            ETIMSK & ~(1 << TOIE3)) {
  return ATM128_POWER_IDLE;
}
// SPI (Radio stack on mica/micaZ
else if (bit_is_set(SPCR, SPE)) {
  return ATM128_POWER_IDLE;
}
// A UART is active
else if ((UCSR0B | UCSR1B) &
    (1 << TXCIE | 1 << RXCIE)) { // UART
  return ATM128_POWER_IDLE;
}
// I2C (Two-wire) is active
else if (bit_is_set(TWCR, TWEN)){
  return ATM128_POWER_IDLE;
}
// ADC is enabled
else if (bit_is_set(ADCSR, ADEN)) {
  return ATM128_POWER_ADC_NR;
}
else {
  return ATM128_POWER_DOWN;
}
</ >
```

Figure 3: Atmega128 Power State calculator code

This result is compared to the actual lowest possible state to which the MCU is allowed to go. Since some external peripherals may require the MCU to not go below a given power state (for example if it will require the CPU soon and the tradeoff between time spent in a low-power state and wake-up latency is not favorable), TinyOS allows them to specify the lowest acceptable power state through the `McuPowerOverride.lowestState()` command.

Despite first appearances, TOSSIM does not offer support for fine tracking of the behaviour of the microcontroller. A port of the `McuSleep` components exists, together with a representation of the hardware registers as entries in a global array of `uint8_t` entries. However, these registers are never modified, and the `McuSleep` component itself is not wired inside the TOSSIM scheduler (which is missing a call to the `sleep()` command when there are no tasks available).

The power state of the microcontroller is of some importance to our battery model: we assume that battery recovery can only take place with the MCU in `POWERSAVE` or `POWERDOWN` modes. Therefore the TOSSIM scheduler is extended to call `McuSleep.sleep()` and we basic tracking of some of the components' state is performed to allow PowerTOSSIM 2 to report meaningful values for the MCU state.

The atm128 implementation also tracks the use of the LED and other ports. LEDs are connected to Port G of the atm128 microcontroller, a 5 bit port. Bits 0, 1 and 2 are used to directly manipulate the LED state, while bit 4 acts as a broadcast bit (controlling all the three LEDs at the same time).

The state of the SPI bus can be derived from the use of the flash and the radio stack (which both use the SPI bus). TOSSIM uses the SimpleFcfsArbiter to emulate the atm128 SPI bus (which allows some tracking of the resources requiring and releasing the SPI bus).

4.3 Peripheral Power Management

TinyOS 2.x divides the power management interfaces into two distinct classes, the microcontroller and the peripherals. A richer set of interfaces (with respect to TinyOS 1.x) is offered for power management of peripherals, described in TEP 115 [8] which can be categorized into two different models: *explicit power management* and *implicit power management*. Our work does not focus on the latter (as used by the at45db flash memory components); simplistically, it can be understood as powering on or off a given device with the explicit power management interface once the resource has been granted (in the case of the at45db interface, the SPI bus). A more detailed description is available in [8].

4.4 Asynchronous interfaces and AT45DB Flash memory

In addition to the `StdControl` (used for peripherals with negligible power-down and power-up times) and `Split-Control` (for peripherals whose power-on/power-down time is of the order of a few milliseconds) interfaces, from TinyOS 1.x, TinyOS 2.x also includes the `AsyncStdControl` interface. This is designed for devices that need to be powered up or down asynchronously, for example inside an interrupt service routine.

All the above interfaces offer a `start()` and a `stop()` command. The `SplitControl` interface, with its *split phase* design, exposes a `startDone()` and a `stopDone()` event, which signals the success of the power up/down call to the component that required it. The calls to those commands happen at a level high enough to be easily tracked inside TOSSIM; thus PowerTOSSIM z can produce a precise trace of the period of use of the peripherals. A detailed description of the impact of peripheral on power use can be gathered from our energy model.

Thanks to the design of TinyOS (especially to the HPL/ HAL/ HIL architecture), PowerTOSSIM z can track a request to perform one of those operations at a relatively high level. We adapt the AT45DB HPL TOSSIM porting code from the one provided by Venkatesh [17] in his port of PowerTOSSIM and incorporate tracking of the erase and CRC operations.

4.5 Chipcon CC2420 radio stack

The most frequently used and energy-intensive peripheral on a mote is the radio. As stated in the TEP 126 - "CC2420 Radio stack" [12] : "The TI/Chipcon CC2420 radio is a complex device, taking care of many of the low-level details of

transmitting and receiving packets through hardware. Specifying the proper behavior of that hardware requires a well defined radio stack implementation. Although much of the functionality is available within the radio chip itself, there are still many factors to consider when implementing a flexible, general radio stack."

The radio stack is divided into multiple layers, each one of these exports, and can use, up to three interfaces: `Send`, `Receive` and `SplitControl`. This is advantageous for tracking the general behaviour of the radio inside TinyOS/ TOSSIM, because even at a very high level, one still has a clear view of the general send and receive commands issued and their relative success. Another advantage is that TOSSIM's radio model is based on the CC2420's values and behaviour [6]. That allowed us to track the radio on, off, send and receive operations directly inside the `tos/lib/tossim/` code (more precisely inside `TossimPacketModelC.nc`), and to benefit from TOSSIM's noise modelling for a more realistic simulation.

Set against this was our difficulty with the CC2420 radio stack's support for a different range of output power transmission levels. An on-mote application can set this by modifying the `TXCTRL.PA_LEVEL` register. Values range from level 3 (-25 dBm) up to level 31 (0 dBm). These have quite different rates of power consumption, from 8.5 mA up to 17.4 mA respectively. Unfortunately there is no support for tracking this change within TOSSIM and this is a major omission for a power consumption simulator. In our implementation, we consider the `PA_LEVEL` to be static and we gather it from the value of `CC2420_DEFAULT_RFPOWER` inside the TinyOS code. If the `-DCC2420_DEFAULT_POWER` flag is not specified at compile time, the default power is set to 31, the maximum. The post-processor behaves similarly.

The CC2420 radio stack implementation features another set of power saving approaches, which are implemented in the Low Power Listening layer. This layer is not compiled in by default and has to be explicitly turned on by setting the `LOW_POWER_LISTENING` variable at compile time. There is no support in TOSSIM (and thus in PowerTOSSIM z) for this layer, though it is of interest for future work on maximizing battery life in applications that do not make heavy use of the radio.

The final issue in PowerTOSSIM z's implementation of the CC2420 radio stack was the device's idle mode. Accordingly to the CC2420 datasheet [3] the chip can enter an `IDLE` mode, where the crystal oscillator is active, but the chip is neither receiving nor sending (Fig 4). There is no support for the idle mode in TinyOS, since, (from post [14]): "No, that state is not implemented in the CC2420 radio stack. It was a conscious decision because the idle state of the CC2420 hardware makes almost no sense from a power management standpoint. You can't do anything in that state and you're burning away almost a milliamp of power."

4.6 Extensibility

The inclusion of support for different physical motes (e.g. mica2 or the newer Telos) would depend on two factors:

- support of their hardware in TOSSIM, and
- the use of correct energy model values (component power consumption, etc.) in the post-processor

The second is just a matter of obtaining the appropriate information from the datasheets and using it in the post-processor, while the first depends on the porting of TinyOS code into TOSSIM. The design of PowerTOSSIM v2.0 is flexible enough to be quickly extended to support new code: it is sufficient to add `dbg` statements any time a specific component behaviour changes. The lower the level at which TinyOS code is ported into TOSSIM, the most accurate the simulation will be.

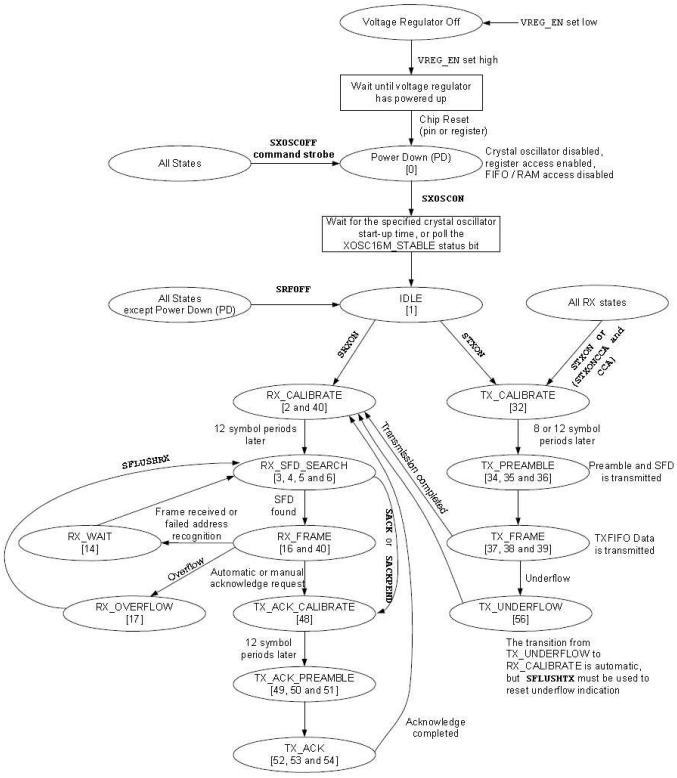

Figure 4: Chipcon CC2420 state diagram (from datasheet). Its IDLE state is not used in TinyOS

5. POWERTOSSIM Z: THE NON-LINEAR ENERGY MODEL

Having described the various devices that PowerTOSSIM z deals with and the details of their implementation, we now present the ideas underlying the design of PowerTOSSIM z and its non-linear energy model. PowerTOSSIM z's architecture resembles that of its MICA2-based TinyOS 1.x predecessor and is divided into two parts, the core and the post-processor. A third module, called PowerCurses, completes the project and offers a ncurses based interface to monitor the state of the motes in the simulation.

Like PowerTOSSIM 1, PowerTOSSIM z is based on a trace model. A number of calls to a central module have been placed in the TOSSIM code (together with some patches to TOSSIM) in the places described in the preceding section. At the lowest level, those calls translate to calls to the `dbg()` debug function of TOSSIM. It is just a matter of connecting the `ENERGY_HANDLER` channel either to the standard output or a file at the start of the python simulation code. The module then generates the traces which capture power usage of the various devices.

The trace in Table 2 shows a mote booting and, later,

Table 2: Sample trace for use by post-processor

```
DEBUG (0): 23542399,LED_STATE,LED2,ON
DEBUG (0): 23542399,LED_STATE,LED1,ON
DEBUG (0): 23542399,LED_STATE,LED0,ON
DEBUG (0): 23542399,LED_STATE,LED0,OFF
DEBUG (0): 23542399,LED_STATE,LED1,OFF
DEBUG (0): 23542399,LED_STATE,LED2,OFF
DEBUG (0): 23542399,CPU_STATE,CPU_POWER_DOWN
DEBUG (0): 23542399,RADIO_STATE,ON
DEBUG (0): 23542399,CPU_STATE,CPU_ACTIVE
DEBUG (0): 23542399,CPU_STATE,CPU_IDLE
[...]

DEBUG (0): 5455507245954,RADIO_STATE,SEND_MESSAGE,
    ON,DEST:65535,SIZE:35,DB:0
DEBUG (0): 5455521894306,RADIO_STATE,SEND_MESSAGE,
    OFF,DEST:65535,SIZE:35
DEBUG (2): 5455521894306,RADIO_STATE,RECV_MESSAGE,
    DONE,DEST:65535
DEBUG (1): 5455521894306,RADIO_STATE,RECV_MESSAGE,
    DONE,DEST:65535
DEBUG (3): 5455521894306,RADIO_STATE,RECV_MESSAGE,
    DONE,DEST:65535
DEBUG (4): 5455521894306,RADIO_STATE,RECV_MESSAGE,
    DONE,DEST:65535
DEBUG (4): 5455521894406,CPU_STATE,CPU_ACTIVE
DEBUG (1): 5455521894406,CPU_STATE,CPU_ACTIVE
DEBUG (2): 5455521894406,CPU_STATE,CPU_ACTIVE
DEBUG (3): 5455521894406,CPU_STATE,CPU_ACTIVE
[...]
```

sending a broadcast message that three other motes receive. The booting up section of this trace provides a good example of TOSSIM's property of executing scheduled events in immediate time. The generated trace can then be passed to the post-processor to estimate the battery energy consumed by the application. This decoupling between trace generation and the battery analysis has several advantages: (i) reducing the amount of changes to the TOSSIM code (essentially just placing command calls at the appropriate position), (ii) simplifying testing with different battery models, (iii) permitting collection of a range of different simulation results and parsing them multiple times with different options without running the simulation every time.

5.1 Battery model post-processor

The post-processor can also work at runtime during the TOSSIM simulation, either receiving the input from `stdin` or, since it is written in python, merging its code with the simulation code and directly connecting the channels. The post-processor has two modes: silent or verbose. In silent mode the whole trace file is analyzed and the final state of the battery is reported at the end. In verbose mode the post-processor outputs a line describing the current state of the battery every time it encounters a line in the trace. Verbose mode is useful if one wants to connect the post-processor to a display application, for example PowerCurses.

PowerCurses can be used to run a graphic simulation remotely without the need of proxying an X Server connection over ssh, or locally without the need of a X based or Windows based system.

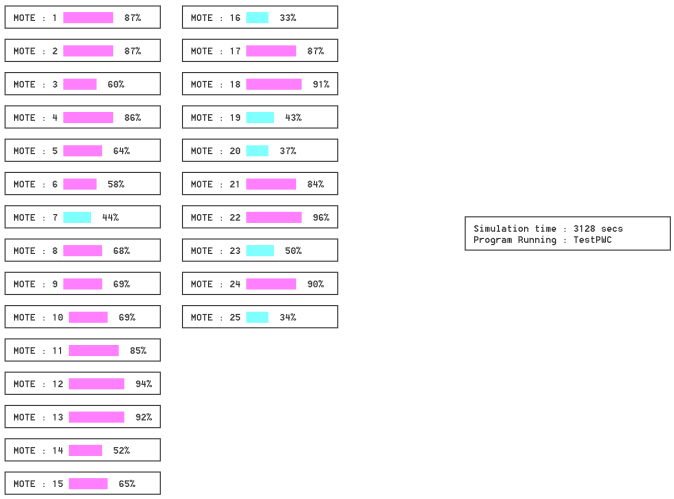

Figure 5: PowerCurses. The ncurses interface shows the actual battery state of the mote on the left and the application name with the actual simulation time on the right

To model the non-linearity of the discharging rate of the battery, additional information is required. We assume the recovery probability varies according to the battery's remaining charge. Following the stochastic approach in [13], we model the probability of recovering a single charge unit per time unit as:

$$p_j(f) = \begin{cases} q_0 e^{-g_N(N-j)-g_C(f)} & f = 0 \\ q_0 e^{-g_N(N-j)-g_C(f)d_f} & f = 1, ..., f_{max} \end{cases} \quad (1)$$

where q_0 is the probability that no current is drawn, N is the battery's total number of charge units, j is the present charge level of the battery in units, d_f is the number of charge units that have been already drained, and the battery's discharge is divided into discrete phases f with $f = 0, ..., f_{max}$. Per [13], g_N is a constant (corresponding to the inverse of the internal resistance of the battery) and g_C is a piecewise constant function of d_f (i.e. the number of charge units already drawn).

The rate capacity effect is also read from a lookup table. This table is interpolated from manufacturer battery measurements [7]. Current demanded is scaled up by a factor that depends on the current itself.

6. EVALUATION

6.1 Known TOSSIM limitations

TOSSIM is a useful simulator, but it was not designed with power tracking in mind. Despite the lack of low level component implementations (which can be fixed by writing additional code), there are a some design issues that affect power estimation for some kind of applications. In particular (i) TOSSIM is a simulator, not and emulator and (ii) within TOSSIM all tasks execute instantaneously.

TOSSIM is a simulator, not an emulator (unlike Avrora [25] or Atemu [15]). It does not capture the behaviour of the mote at the instruction level. Thus, if an application is CPU intensive, it is very hard to capture the exact amount of time the MCU spends in ACTIVE state. PowerTOSSIM

1 tried to solve this problem using a basic block accounting mechanism [23]: the application binary is divided into basic blocks (sequences of instructions without a branch), each occurrence of each basic block is counted and then CPU cycle calculations are applied to each basic block and multiplied by the times it was executed.

Occasionally this method can lead to inaccurate results [20]. If precise energy estimation is needed for a CPU intensive application the best solution may be to use one of the previous cited emulators and merge the results. Moreover, the energy model described herein can be relatively easily used in conjunction with other power consumption analyzers, especially if they use a trace based system.

Within TOSSIM, all the tasks execute in instantaneous time. Thus, for example, interrupts can not fire at an arbitrary time and preempt a running task. While this is indeed a minor issue, it can lead to incorrect results for some applications. Once again, if this is the case, the solution is to use an emulator and forsake the speed advantages of TOSSIM.

Based on the discussion so far, emulators appear to be the a best solution for tracking a mote's power consumption, so why still use TOSSIM and create a port of PowerTOSSIM? First of all, TOSSIM is fast and highly scalable, a desired characteristic if one wants to repeat multiple test changing some configuration. Moreover, TOSSIM maps directly inside TinyOS code and is simple. Testing code can be rapidly developed in Python. All these properties have made it the testing environment of choice for TinyOS applications. Having a common testing environment permits users to share their results and, as many open source projects demonstrate, increases the number of patches and extensions arriving from the community, thus improving the overall quality of the code.

6.2 Results

In this section we contrast results obtained from PowerTOSSIM v1.x for *CntToLedsAndRfm* from TinyOS 1 (on a MICA2 mote) and from PowerTOSSIM Z for the functionally similar application *RadioCountToLeds* from TinyOS 2 (on a MICAz). The application periodically emits a packet while cycling through the LEDs.

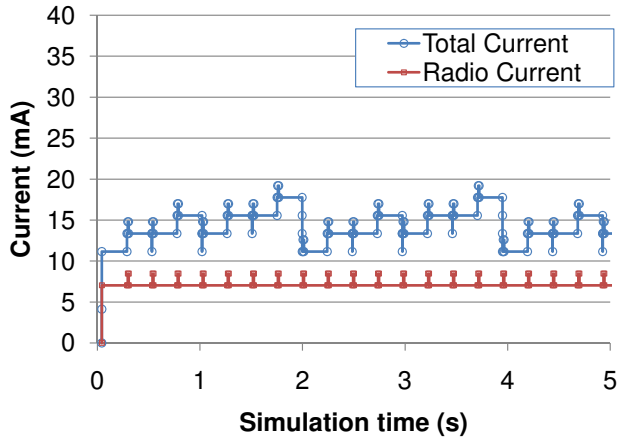

Figure 6: Trace of TinyOS application *CntToLedsAndRfm* for PowerTOSSIM v1.x

Figs. 6 and 7 show a trace of the current, in which the increased peak power consumption of the MICAz's new radio stack is visible.

The radio receive current is now higher than the transmit current, seen as periodic short drops in the radio current in Fig. 7.

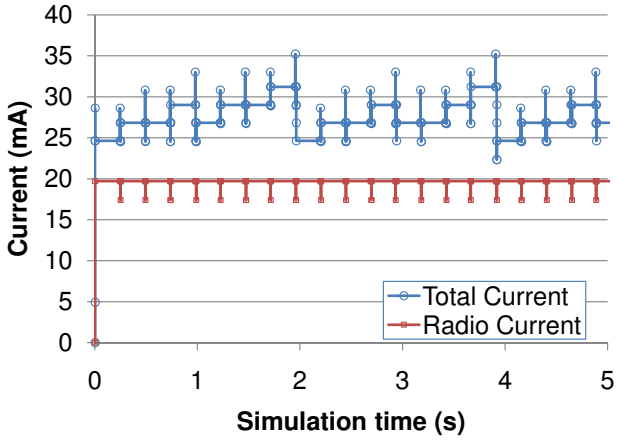

Figure 7: Trace of TinyOS application *RadioCountToLeds* for PowerTOSSIM Z

The increased power consumption of the MICAz mote in this case further emphasises the importance of using the hardware's low-power modes when appropriate. The MICAz's radio has a lower per-bit power rate, but the gain is not realized if the radio is left in receive mode when not transmitting.

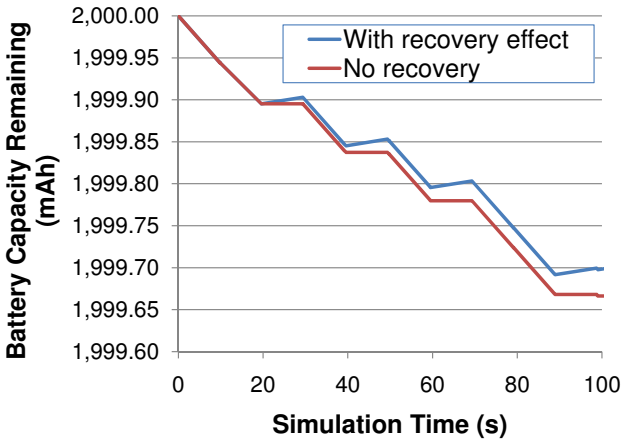

Figure 8: Recovery effect. Shaded area indicates mote is powered down

To illustrate the recovery effect of the model, it is not possible to use the *CntToLedsAndRfm* application, since its current demand never drops low enough to allow recovery. Instead we simulate a custom application, adding periods of 'dead time' in which the mote is powered down. This allows the current to drop to zero and hence the battery to recover. Fig 8 shows, in a qualitative way, how the inclusion of three

short, ten second periods of "recovery time" in an 90 second time interval impacts on the remaining battery capacity.

7. CONCLUSION

The energy-constrained nature of wireless sensor networks makes the simulation of power consumption an important and active research topic. The continuing development of sensor mote technology, with new models regularly coming to market, means that any simulator must be flexible and easily extensible. TinyOS and TOSSIM look set to continue their popularity as the platform of choice for sensor networks. PowerTOSSIM was a successful implementation of power simulation of TinyOS v1.x; we have extended the port of PowerTOSSIM v2.0 for the MICAz mote. We have implemented a flexible battery model that captures the non-linear behaviour of modern batteries.

Possible future adaptions to be made to TOSSIM include the completion of support for low power states and the passing of radio transmission power information to the post-processor. Calibration of the battery model for different battery types and inclusion of the kinetic battery model of [18] would also give rise to improved performance.

8. ACKNOWLEDGMENTS

This publication has emanated from research conducted with the financial support of Science Foundation Ireland.

9. REFERENCES

[1] ATM128 datasheet, Jan. 2007. Available at `http://www.siphec.com/microcontroller/ATmega128(L)_complete.pdf`.

[2] A. Barberis, L. Barboni, and M. Valle. Evaluating energy consumption in wireless sensor networks applications. *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, 2007.

[3] CC2420 datasheet, Jan. 2008. Available at `http://inst.eecs.berkeley.edu/~cs150/Documents/CC2420.pdf`.

[4] C. F. Chiasserini and R. R. Rao. A model for battery pulsed discharge with recovery effect. In *Proceedings of IEEE WCNC, vol. 2*, pages 636–639, 1999.

[5] C.-Y. Chong. Sensor Networks: Evolution, Opportunities,and Challenges. In *Proceedings of the IEEE, vol. 91, no. 8*, Aug. 2003.

[6] T. Community. Lesson 11 - tossim, Jan. 2008. Available at `http://docs.tinyos.net/index.php/TOSSIM`.

[7] Product Datasheet - Energizer E91. Available at `http://data.energizer.com/PDFs/e91.pdf`.

[8] K. Klues, V. Handziski, J.-H. Hauer, and P. Levis. Power management of non-virtualized devices. Available at `http://www.tinyos.net/tinyos-2.x/doc/txt/tep115.txt`.

[9] F. Lewis. Smart Environments - Wireless Sensor Networks, 2004. Available at `http://arri.uta.edu/acs/networks/WirelessSensorNetChap04.pdf`.

[10] MICA2 datasheet, Jan. 2008. Available at `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`.

[11] MICAz datasheet, Jan. 2008. Available at `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`.

[12] D. Moss, J. Hui, P. Levis, and J. I. Choi. Cc2420 radio stack, June 2007. Available at `http://www.tinyos.net/tinyos-2.x/doc/html/tep126.html`.

[13] D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan, and K. Lahiri. Battery life estimation of mobile embedded systems. In *14th International Conference on VLSI Design (VLSID 2001)*, pages 55–63, 2001.

[14] J. Polastre. Cc2420 idle state [tinyos-help] mailing list post, Dec. 2005. Available at `http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2005-December/013742.html`.

[15] J. Polley, D. Blazakis, J. Mcgee, D. Rusk, and J. S. Baras. Atemu: a fine-grained sensor network simulator. *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pages 145–152, 2004.

[16] PowerTOSSIM website, Jan. 2008. Available at `http://www.eecs.harvard.edu/~shnayder/ptossim/`.

[17] T. V. Prabhakar, S. Venkatesh, M. S. Sujay, K. Joy, and K. Praveen. Simulation blocks for tossim-t2. In *WISARD 2008, A workshop in COMSWARE 2008*, 2008.

[18] V. Rao, G. Singhal, A. Kumar, and N. Navet. Stochastic battery model for embedded systems. In *VLSI Design, 2005. 18th International Conference on*, 2005.

[19] D. Schmidt, M. Kramer, T. Kuhn, and N. Wehn. Energy modelling in sensor networks. *Advances in Radio Science*, 2007.

[20] 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*, Nov. 2004.

[21] The SPICE website, Jan. 2008. Available at `http://bwrc.eecs.berkeley.edu/Classes/IcBook SPICE/`.

[22] R. Szewczyk, P. Levis, M. Turon, L. Nachman, P. Buonadonna, and V. Handziski. Microcontroller power management, Jan. 2007. Available at `http://www.tinyos.net/tinyos-2.x/doc/html/tep112.html`.

[23] T. K. Tan, A. Raghunathan, G. Lakshminarayana, and N. K. Jha. High-level software energy macro-modeling. In *Design Automation Conference*, pages 605–610, 2001.

[24] TinyOS website. Available at `http://www.tinyos.net/`

[25] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. *Proc 4th International Symposium on Information Processing in Sensor Networks, 2005. IPSN 2005.*, pages 477–482, 2005.

[26] TOSSIM website. Available at `http://www.cs.berkeley.edu/~pal/research/tossim.html`.