Simulating Heterogeneous and Larger-Scale Wireless Sensor Networks with TOSSIM TinyOS Emulator

Jiakai Li, Gursel Serpen

Abstract

TOSSIM is a wireless sensor network (WSN) emulator which is useful for assessing and evaluating application code in TinyOS operating system and Mica mote based environments. Although TOSSIM provides a high-fidelity and very accurate emulation environment for WSN applications, there are some significant drawbacks that might hinder simulation of larger scale WSNs. TOSSIM can simulate WSNs with up to 1,000 motes under certain conditions. Another fundamental shortcoming of TOSSIM is that it does not support heterogeneous networks, although this issue originates as a result of a limitation imposed by TinyOS, where every mote must run the same code. In furtherance, the TOSSIM memory space allocation for the application program running on motes appears to be fixed regardless of the number of motes in the WSN. This leads to having to write smaller application for a mote as the mote count increases. Finally, because of its bit-level granularity, TOSSIM simulation time increases quickly to several weeks for larger-scale sensor networks with around 1000 motes. We propose a number of suggestions and workarounds to facilitate larger-scale WSNs to be simulated with reasonably-functional application code and within quicker simulation time periods in the TOSSIM environment. In order to simulate heterogeneous networks, a simple If-Then control structure is proposed for the application code. Packet-TOSSIM is an option to reduce the simulation time if bit-level granularity is not required. The nesC application code must be compact and optimized for size for larger mote counts so that the entire simulation can fit into the TOSSIM process memory space.

Keywords: Wireless Sensor Networks, TinyOs-nesC, TOSSIM, Large Scale Simulation, Heterogeneous Sensor Network

1. Introduction

Wireless sensor networks (WSN) are an emergent and promising technology with wide range of new and novel applications from agriculture to medicine to monitoring of habitats and environment or man-made structures such as tall building or bridges. A wireless sensor network is composed of wireless nodes each of which has on board microcontroller, radio, sensors and power unit. The number of such nodes or motes can be on the order of tens, if not hundreds, of thousands for some applications. Accordingly, prototyping such a large-scale and distributed system is not feasible and often simulation (or emulation) is the final step prior to the actual deployment in the field. Therefore, simulators are essential for research and development of WSNs, and offer the only feasible choice for testing new applications and protocols.
TOSSIM is a bit-level emulator for wireless sensor networks [1] and intended for TinyOS-Mica platforms. TinyOS [2] is a lightweight operating system specifically developed for WSNs while Mica is a family of motes [3] or actual sensor nodes. Application code for sensor networks is written using the nesC (network embedded systems C) programming language for TinyOS environments. TOSSIM is compiled directly from TinyOS-nesC code and runs natively on a desktop computer. TOSSIM can simulate up to a thousand sensor nodes or motes under certain conditions. However, it does place a restriction in that every mote in a simulation needs to run the same nesC code which is employed to develop the application code. This makes TOSSIM not applicable for heterogeneous networks. The TOSSIM memory space allocation for the application program running on motes appears to be fixed regardless of the number of motes in the WSN. This leads to having to write smaller application code for a mote as the mote count increases. Finally, because of its bit-level granularity, TOSSIM simulation time increases quickly to several weeks for large-scale sensor networks, which may have 1000 or more motes.

The goal of this paper is to provide suggestions and workarounds to facilitate larger-scale WSNs to be simulated within quicker simulation time periods in the TOSSIM environment. In order to simulate heterogeneous networks, a simple If-Then structure is proposed for the application code. Packet-TOSSIM is an option to reduce the simulation time if bit-level granularity is not required. The nesC application code must be compact and optimized for size for larger mote counts so that the entire simulation can fit into the TOSSIM process memory space. Further details on all these points will be presented in the following sections.

2. TinyOS and nesC

TinyOS is an open-source operating system for wireless sensor networks and available under the BSD-license [1]. nesC (network embedded systems C) is a component-based, event-driven programming language. It is used to build application programs for the TinyOS-Mica platform. nesC is built as an extension to the C programming language. In nesC, components are "wired" together to build applications for the TinyOS environment. TinyOS itself is written in nesC as a set of cooperating tasks and processes. More specifically, all the system components, libraries, and applications in TinyOS are written in nesC. Many system-defined interfaces and components are provided for common abstractions such as packet communication, routing, sensing, actuation and storage.

3. The TOSSIM emulator

TOSSIM is a discrete event emulator for the execution of nesC model on TinyOS-Mica hardware [1]. In TOSSIM, an event is generated for each transmitted or received bit or packet. TOSSIM further functions as an emulator of actual hardware through mapping hardware interrupts to discrete events. TOSSIM also has a built-in simulated radio model. As such, TOSSIM is a high fidelity simulator and emulator of a network of TinyOS-Mica motes. Emulated hardware components are compiled together with real TinyOS components using the nesC compiler. Thus, an executable with real TinyOS applications over a simulated physical layer is obtained. Additionally, there are also several communication services that provide a way to feed data from external sources.

TOSSIM can simulate at the bit (TOSSIM or TOSSIM-bit) or packet level (TOSSIM-packet), which is a user-selectable feature. TOSSIM-packet is a drop-in replacement for TOSSIM's radio stack that operates at the packet, rather than the bit level. Preliminary testing results indicated that it performs about 100 times faster than the default bit-level radio stack [4]. Packet level simulation does not capture the subtleties and behaviour of the bit-level simulation, but it is useful for testing and simulating larger networks. The packet-level simulation is compatible with TOSSIM's loss radio models. TOSSIM-packet does not model packet collisions (e.g., the hidden terminal problem), MAC contention, or variable transmission time due to these effects or packet length. Packet transmission takes 1/40th of a second, based on the empirical observation that the Mica radio stack can handle about 40 packets per second. Some preliminary tests indicate that packet loss rates due to bit errors from the packet-level model are roughly equal to those in the bit-level model [4].

TOSSIM’s ability to simulate large mote counts is primarily limited by its bit-level or packet-level granularity: its performance degrades as the communication traffic increases. Channel sampling is also simulated at the bit or packet level and consequently the use of a Carrier Sense Medium Access (CSMA) protocol causes significant overhead. In the TOSSIM environment, the wireless communications is assumed to be error-free for any mote within the one-hop neighbourhood of a given mote. In other words, every bit transmitted in the network is received...
without any error. All signals are of equal strength, and collisions are modelled through the use of a CSMA protocol and there is no cancellation during network communication. This means that distance does not affect the signal strength. Motes in the network may hear the overlap of the signals.

In TOSSIM default compile-time upper bound for the number of motes in a wireless sensor network is set to be 1000 as indicated by the line of code in the file <nido.h>[5] presented in Figure 1. We installed TOSSIM within Cygwin (the Linux emulator on top of Windows XP) and on the Linux OS. However, the installation was successful only on the 32-bit systems. In this context, our own experience also indicates that 1000 motes also turned out to be the pragmatic upper bound since WSNs with more motes, such as those in the neighbourhood of 1200, required more than a month of simulation time and often led to TOSSIM process to exit prematurely. We also observed a relationship between the size of the application program running on a mote and the number of motes that could be simulated in TOSSIM. Specifically, as the size of the nesC application program and its memory allocation requirements due to static arrays got larger it had an adverse effect on the number of motes for the WSN. To accommodate the larger size application programs, the number of motes had to be decreased for the TOSSIM process to run to completion. One likely explanation is that the TOSSIM process itself has a fixed memory allocation by the operating system. Then, as the number of motes in the WSN increases, this leads to having to write smaller application program for a mote since all the static arrays specific to a mote must be created individually and separately for each mote in the WSN.

```
...  
 ifndef NIDO_H_INCLUDED  
 #define NIDO_H_INCLUDED  
 enum {  
   TOSNODES = 1000,  
   DEFAULT_EEPROM_SIZE = (512 * 1024) // 512 KB  
 };  
 enum {  
   TOSSIM_RADIO_MODEL_SIMPLE = 0,  
   TOSSIM_RADIO_MODEL_LOSSY = 1,  
   TOSSIM_RADIO_MODEL_PACKET = 2  
 };  
...  
```

Figure 1. “TOSNODES” indicates the upper bound for the number of motes in the file “nido.h”

TOSSIM is intended for simulating homogeneous sensor networks. In other words, every node must run the same code. Therefore, TOSSIM cannot be used to evaluate heterogeneous WSN applications where motes must necessarily execute different application programs unless a workaround is formulated to overcome this apparent limitation.

4. Simulating heterogeneous and larger-scale WSNs in TOSSIM

We have used TOSSIM to simulate a wireless sensor network where each mote was embedded with a single neuron and its computation model [6]. The neurons collectively formed a Hopfield recurrent neural network configured as a static optimizer [7-9] to solve a graph theoretic optimization problem, namely the minimum weakly-connected dominating set. We have simulated instances of WSNs with up to 1200 motes. Each mote implemented the computational model of a single neuron and stored all the needed data associated with those calculations including, but not limited to, output values for all the other neurons to which it is connected to, its weight vector, other parameters of interest. Each mote implemented requisite computations asynchronously and the collection of computations over the entire sensor network or the equivalent neural network took place in parallel and distributed mode. The associated simulation study establishes the basis for the suggestions proposed in this paper to facilitate simulations of larger-scale and heterogeneous WSNs using the TOSSIM emulator.
4.1 Simulating a larger-scale network

By default, although the standard TinyOS distribution sets the compile-time upper bound of network size (the maximum numbers of nodes) to be 1000, we can change this value by specifying “TOSNODES” in nido.h. However, “TOSNODES” is not the only parameter that determines the network size that can be simulated in TOSSIM environment. The memory space allocation requirements of nesC application program to execute on each mote, such as those through static array declarations, as well as the size of the application program itself also play a role for the number of motes that can be simulated. For example, Simulation of a WSN in TOSSIM, where each mote executes one of the simplest nesC applications - Blink [10] which has only about 60 lines in the module file and, more importantly, no noteworthy static memory allocation, can be accomplished for up to about 9000 motes. nesC source code for Blink application is presented in Figure 2 while pointing out the fact that the code does not utilize any significant memory allocation. However, simulating a more complicated nesC application in TOSSIM, which is the case study referred to herein is possible for only about up to 1200 motes. This application has more than 800 lines in the module file, but, more importantly, four static arrays where each has a size equal to the number of motes or nodes in the WSN as shown in Figure 3. It is therefore critical to minimize the static memory allocation requirements of the nesC application designated to execute on motes if it is desirable to maximize the number of motes for the WSN being simulated in TOSSIM.

```
// File: Blink.nc
configuration Blink {
}

implementation {
    components Main, BlinkM, SingleTimer, LedsC;

    Main.StdControl -> BlinkM.StdControl;
    Main.StdControl -> SingleTimer.StdControl;
    BlinkM.Timer -> SingleTimer.Timer;
    BlinkM.Leds -> LedsC;
}

// File: BlinkM.nc
module BlinkM {
    provides {
        interface StdControl; }
    uses {
        interface Timer;
        interface Leds; }

    implementation {
        command result_t StdControl.init() {
            call Leds.init();
            return SUCCESS; }
        command result_t StdControl.start() {
            return call Timer.start(TIMER_REPEAT, 1000); }
        command result_t StdControl.stop() {
            return call Timer.stop(); }
        event result_t Timer.fired() {
            call Leds.redToggle();
            return SUCCESS; }
    }
}
```

Figure 2. nesC source code for configuration and module files for Blink application
5.2 Simulating heterogeneous networks

In order to simulate heterogeneous networks in TOSSIM, a simple If-Then structure is proposed for the application code as exemplified in Figure 4. Assume that there are two kinds of motes in the wireless sensor network, the supervisory mote and generic motes. Generic motes perform computations while the supervisory mote acts as a “supervisor” – it monitors, collects, processes and broadcasts the global information related to computations. The supervisory mote may be responsible broadcasting start and end times of different phases of computations and may communicate the start and end times of all such phases to the generic motes. The If-Then structure in Figure 4 shows how to implement the functionality for each type of mote. If the types of motes is more than two, then one can extend this implementation through an If-ElseIf-ElseIf-….-Else template as well.

```
result_t Command() {
    if (SupervisoryMote == TRUE) {
        // This code segment is executed by the supervisory mote only!
        ...
    } else {
        // This code segment is executed by a generic mote only!
        ...
    }
}
```

Figure 4. If-Then structure for nesC code implementation for heterogeneous networks

In TOSSIM, each mote is assigned a unique ID. During the initialization, each mote in the network first confirms its own ID in TOSSIM. Any ID number can be designated to be the supervisory mote although in our case we set the supervisory mote to be the one with the ID of 0. Therefore, during initialization, all the motes check their own IDs to be 0 or not. Once a mote determines that its ID is 0, it will mark itself as the supervisory mote in its local memory. Other motes which will have an ID other than 0 will mark themselves as the generic motes in their local memories. Consequently, each mote knows its role in the network once the initialization is complete. When a mote wakes up to attend its periodic business, it will first check to see if it is the supervisory mote or generic mote. Upon determination of this aspect, it will execute the corresponding part of the code in the If-Then structure. This will be the If part if the mote is supervisor, or the Then part if the mote is generic.

5.3 Faster simulation in the TOSSIM environment

To perform quicker simulations in the TOSSIM environment, one can use TOSSIM-packet [4] instead of the default TOSSIM-bit. TOSSIM-packet is a drop-in replacement for TOSSIM’s radio stack that operates at the packet, rather than the bit level. If bit-level operations and behaviour are not of interest, then TOSSIM-packet can be employed to test the larger networks with significant speedup in overall simulation time compared to the TOSSIM-bit. TOSSIM-packet can be downloaded at http://www.tinyos.net/tinyos-1.x/tos/platform/pc/packet/. To use TOSSIM-packet, it is necessary to create a local Makefile file in the application, and add a flag named

```
“PFLAGS” with the value of “-I$(BASEDIR)/opt/tinyos-1.x/tos/platform/pc/packet” in the same Makefile. Next, the entire application needs to be rebuilt in TOSSIM with the “make pc” command line specification. This will make it possible to execute simulations faster with the TOSSIM-packet.

5. Conclusions

TOSSIM is a bit- or packet-level emulator of wireless sensor networks based on TinyOS-Mica platform and is native to the Linux or Cygwin on Windows operating systems. Simulating larger scale wireless sensor networks with mote counts reaching 1000 requires special arrangements and optimizations to the nesC application code and the simulation environment. This paper, proposed suggestions and workarounds to facilitate larger-scale WSNs to be simulated in the TOSSIM environment. In order to simulate heterogeneous networks, a simple If-Then structure is proposed for the application code. Packet-TOSSIM was suggested as an option to reduce the simulation time if bit-level granularity is not required. The nesC application code must be compact and optimized for its memory allocation requirement for larger mote counts so that the entire simulation can fit into the TOSSIM process memory space.

References