Assessing Time Complexity of Applications for TinyOS-Mica Wireless Sensor Networks in TOSSIM Emulator

Gursel Serpen, Jiakai Li

Electrical Engineering and Computer Science, University of Toledo, Toledo, Ohio 43606, USA

Abstract

We propose an approach to assess and profile the time complexity of a wireless sensor network application executing in the TOSSIM environment since TOSSIM lacks the built-in facilities to determine the run time for an application that is intended for execution in parallel and distributed mode on real sensor network hardware. This procedure, which is conceived to estimate the time complexity in approximate terms, requires counting the number of timer periods needed to execute certain functionality of an application. Consequently, the length of a time interval associated with the sleep-wakeup periods for motes is used to approximate the time complexity of the application. We present the application of the proposed methodology for simulation of a wireless sensor network application in the TOSSIM environment.

Keywords: Wireless sensor networks, time complexity, TOSSIM, TinyOS-nesC, artificial neural network, Hopfield neural network, optimization

1. Introduction

Wireless sensor networks (WSN) are an emerging technology due to recent advancements in very small-scale manufacturability and high-scale integration of various electronic components in a single packaging [1]. A WSN may consist of hundreds or thousands of sensor nodes (or motes), each of which is a standalone packaging of electronics necessary to hold a number of sensors, a CPU-based miniaturized computing platform, a depletable or rechargeable power unit, and a radio trans-receiver as well as an antenna as its core.

Current and projected applications of wireless sensor networks encompass a wide variety of domains which have been traditionally challenging to access due to many reasons including potential harm to humans, being at remote sites or distributed over very large areas, and being subject to harsh geo or meteorological circumstances among others [2]. Wireless sensor networks are conceived to be deployed and expected to operate autonomously and particularly in non-hospitable environments without human involvement. Given that a WSN is a complex and large-scale system, which could consist of hundreds or thousands of motes, in many cases, it is impractical to experiment on real WSN systems since deploying, managing, debugging and testing a large number of sensor motes is a very challenging task. Some applications require specific deployment scenarios (e.g. dangerous areas). Therefore, use of simulators is essential to develop and prototype WSNs, and offers a desirable approach for testing new applications and protocols.
TOSSIM [4] is a hardware-level emulator of WSNs based on TinyOS operating system [9] and Mica mote platform [3]. It is compiled directly from the TinyOS-nesC code. The TOSSIM simulator runs natively on a desktop computer, and comes in two flavors: it can emulate events at the bit or packet level depending on the user need. It offers a high-fidelity and realistic simulation platform for up to 1000 motes at the expense of extended simulation times. Although it is an emulator, TOSSIM lacks the facilities to track the execution time of an application that would typically run in parallel and distributed mode on a physical wireless sensor network. This is certainly a significant shortcoming of this simulator. An extension of the TOSSIM emulator which is called TimeTOSSIM had been proposed in [10,11]. TimeTOSSIM counts the number of clock cycles for each of the instructions in the application executable to assess the execution time for that application. It was reported that TimeTOSSIM was up to 10 times slower than TOSSIM for the simulation study conducted in [10]. Another WSN emulator called Avrora [12] appears to perform precise timing measurements and yet was reported to be up to 100 times slower than TOSSIM [10,11].

For very large scale WSNs, this could pose problems for the needed simulation time for the associated application. Consequently, it may be desirable to perform execution time measurements in a coarser scale without causing the TOSSIM simulator to slow down noticeably. The goal of this paper is to provide an approach to facilitate assessment and profiling of the time complexity of an application executing in the TOSSIM environment with negligible effect on the simulation time.

2. Background

TinyOS is the most widely used operating system designed for distributed wireless sensor networks [9]: it is open source and governed by the BSD-license. nesC (network embedded systems C) is a component-based, event-driven programming language used to build applications for the TinyOS platform [5]. nesC is built as an extension to the C programming language with components "wired" together to run applications on TinyOS. TinyOS is written in nesC as a set of cooperating tasks and processes.

The MICA mote platform is a third generation device used for enabling low-power, wireless sensor networks available in 2.4 GHz and 868/916 MHz [3]. There are two kinds of MICA motes with similar design. The MICAz mote is using a IEEE/ZigBee 802.15.4 board with 2.4 GHz technology and the MICA2 has an 868/916 MHz multi-channel radio transceiver designed for low-power, wireless sensor networks. The MICA mote platforms are fully compatible with the TinyOS/nesC software framework and enable users to set up ad-hoc wireless sensor network hardware prototypes with relative ease.

The TOSSIM simulator does not model the execution time for any code segment: in other words, from TOSSIM’s perspective, a piece of code runs instantaneously [4]. The implication of this is that it is implicitly assumed that any mote is able to complete the execution of the application code during the time period when it is awake. TOSSIM process execution time is the actual simulation time acquired from the operating system for the TOSSIM process itself. In other words, it is the real or wall clock time from the simulation start (mote deployment) to simulation end. This wall clock (simulation) time is also highly related to the simulation platform: the operating system, memory and CPU configuration. Since there are typically no more than several CPUs or cores on a von Neumann based workstation, the computations for emulating motes within the TOSSIM environment cannot be considered parallel. The real simulation time for the TOSSIM process itself as reported by the host operating system is not appropriate, and thus cannot be used for time complexity measurement for an application program, which is supposed to execute in parallel and distributed mode across the entire wireless sensor network.

The time related measurements and parameters available in the TOSSIM environment include “sim_time” and so-called simulation time ticks [4]. The “sim_time” parameter value is available through the TOSSIM GUI which is called the TINYVIZ. It returns the number of (virtual) CPU ticks spent since the beginning of a simulation. Since time is kept at a 4 MHz (4 × 10^6 ticks/second) granularity (the CPU clock rate of the Mica mote based platforms), the timer events are four million ticks apart, and the 50 μs ADC capture takes 200 ticks between a request and interrupt. “sim_time” will not change until events happen. Its value is only increased upon various events, and the incremented value is determined by respective models (i.e. tos/lib/tossim/*Model*). For example, the increment of “sim_time” caused by a packet transmission is determined by the packet model (as explained in the nesC source file TossimPacketModelC.nc). It does not care for the size of a packet, the processing time, the delay and so on. For some cases, if “sim_time” does not increase for a while, it means the application hasn’t triggered any “event” that increases “sim_time” so far. Consequently, the “sim_time” parameter is not useful for the purpose of actually measuring the time complexity in some meaningful sense since it does not model the simulation time of the network.
computation for the application program. Instead, it captures the number of events and how these events affect the CPU time.

3. Proposed Methodology

The basic idea for an approximation to the time complexity of an application executing on a WSN platform in the TOSSIM environment is based on counting the number of timer firings throughout the entire simulation period and relating this overall count to time complexity of the same application. In a typical WSN, each mote has a dedicated timer embedded which is mainly used for waking up the mote for tasks designated for periodic processing. The length of time periods is determined based on a set of considerations including but not limited to application code execution requirements, stored energy consumption, communication requirements etc. It will be assumed that for a given application the timer firing period is fixed for any size network as measured in terms of its mote count. Under this assumption, timer firing counts can be employed as a measure of the time complexity.

3.1 Case study

A hierarchical infrastructure for wireless sensor networks will be employed as a case study to demonstrate the proposed methodology for computation of the time complexity of an application. The case study entails computing solutions for the weakly connected minimum dominating set (WCDS) for a graph [6,7], which represents the wireless sensor network and its wireless connectivity topology. Solution computation is formulated as a static optimization problem, and its solution is searched for through the Hopfield recurrent neural network algorithm. The Hopfield neural network [8], which is configured as a static optimizer to search for a solution of the minimum WCDS problem. Furthermore, the Hopfield network optimizer is mapped to the wireless sensor network in fully parallel and distributed manner where each mote is embedded with a single neuron and its computation model.

3.1.1 Minimum WCDS problem

A dominating set (DS) is a subset of nodes or vertices in a graph (a typical abstract model of a sensor network) such that each node is either in DS or has a neighbour in DS [6]. A connected dominating set (CDS) is a connected DS, that is, there is a path between any two nodes in CDS which does not use nodes that are not in CDS. It might be favourable to have few nodes in the CDS. This is known as finding the minimum connected dominating set (MCDS) in an arbitrary graph. More formally, the minimum CDS problem is defined as follows: given an (arbitrary) undirected graph find a CDS with a minimum number of nodes. The minimal weakly connected dominating set problem is of interest since it is comparably easier to compute. A subset of nodes $S$ in a graph $G=(V,E)$ is weakly connected if the weakly induced subset $S_w$ is connected; $S_w$ consists of $S$ and all the neighbours of $S$ while the edges in $S_w$ are from $E$ where each has at least on endpoint in $S$. There are number of challenges to face for WSN implementations given that computing the minimal WCDS is NP-hard. Accordingly, finding a WCDS that is "close" to the minimum (i.e. an approximation rather than an exact solution) might be desirable in most cases. Furthermore, the solution must be local since global solutions are impractical for dynamic distributed networks. A minimum WCDS based on the graph model of a wireless sensor network is the primary underlying framework for the network infrastructure.

3.1.2 WSN-ANN architecture

Hopfield network offers a true “real-time” distributed optimization algorithm for computation of a local optimum solution of a static optimization problem for a hardware-centric implementation that takes advantage of the high-degree of inherent parallelism. The promise is a quick and local optimum solution, and scalability of the computation time with the increase in the size of the problem [8]. The minimum WCDS problem [6,7] can be mapped to the Hopfield network dynamics as follows. Assume a graph has a set of $N$ vertices, $V_i, i=1,2,...,N$, and up to $N^2$ edges, $e_{ij}, i,j=1,2,...,N$, where some of the edges may not exist. Consider a Hopfield neural network with $N$ neurons where outputs of neurons are represented by $z_1,...,z_N$. Each neuron in the neural network will be mapped or correspond to a vertex in the graph. An active neuron ($z_i=1$) will represent that the vertex to which it is mapped is selected for inclusion in the dominating set. All other neurons whose corresponding vertices in the graph have an edge to the vertex mapped to this active neuron should be inactive ($z_j=0$). Also for any neuron that is inactive, exactly one neuron should be active among all neurons which represent its adjacent vertices. These statements can be captured by the following error function under the assumption that neuron output values converge to limiting values in the interval $[0, 1]$:
where it is required that $g_a, g_b \in R^+$. The error term has a globally minimum value of zero when both constraints are satisfied or, equivalently stated, when both terms assume a value of zero. The first term has a minimum value of zero when all adjacent neurons of an active neuron are inactive. The second term is zero when exactly one neuron is active among all the adjacent neurons for a given inactive neuron. This error function can be associated with the generic Liapunov function for Hopfield dynamics to derive values for the weights, biases and threshold for each of the neurons in the network, which is then considered to have been configured to solve the minimum WCDS problem. Further elaboration for solving the minimum WCDS problem with a Hopfield recurrent neural network embedded into a WSN is presented in [13].

The procedure of embedding the Hopfield neural network within the wireless sensor network, which serves as the parallel and distributed hardware realization, is relatively straightforward as described next. Assume a wireless sensor network with $N$ nodes (motes). Each WSN mote is assigned a single (Hopfield net) neuron: each mote computationally implements a single neuron (i.e. calculates the $k$-th iteration value for the discrete-time equivalent of the dynamics equations) along with the storage needed for the weight vector for the neuron, bias and threshold, nonlinearity slope, and others. The weight vector, and bias and threshold terms for a given neuron residing on a given mote are initialized to the values for weights, biases and thresholds. In general, any given neuron can talk to any other neuron in the network (through multi-hop communications over the WSN in many cases), and thus establishing the required connectivity of the Hopfield neural network as dictated by the specific optimization problem energy function. Connections to neurons on one-hop neighbor motes as dictated by the current trans-receiver range settings will be direct or without any intermediaries. Connections to neurons residing on motes that are not one-hop neighbors of the current mote will be over multiple hops. In the case of the minimum WCDS problem, each neuron will need to receive inputs from those neurons residing on motes that are one-hop neighbors for the mote that is the host as indicated by the energy function formulation in Equation 1.

3.1.3 Time complexity measurement

The basic idea is counting the number of timer periods needed to converge to a solution by the Hopfield neural network. This measure can be used in conjunction with the length of the time interval associated with the sleep-wakeup periods, which is called the time interval ($t_i$), to assess the time complexity of an application. There are two time related measurements for each simulation scenario: one measurement is related to counting the total number of timer firings ($TC_{total}$), and the other parameter is the number of timer periods needed for neural network computations ($TC_{neuro-comp}$).

Hopfield network dynamics may demonstrate substantial variations in terms of computational effort to accomplish the convergence to a fixed point as measured by the number of relaxation count. In other words, the real simulation time is not comparable directly among different searches conducted by the same Hopfield network: some cases may take only one relaxation, while others may take several relaxations to locate a local minimum solution. This variation is fundamentally affected by the random initial values of neuron outputs, and the random update order of neurons for asynchronous computation mode. Accordingly, the time measurement of interest is “normalized” time ($T_{normalized}$). We used the following approach to normalize the raw simulation data. The parameters measured are total number of timer ticks for a solution to be computed ($TC_{total}$), total number of timer ticks for neural network computations ($TC_{neuro-comp}$), and the number of relaxations for the Hopfield network ($R$). Therefore, the number of timer ticks or firings for processing not related to neuro-computation is given by $TC_{total} - TC_{neuro-comp}$. Average number of timer ticks for neuro-computation in a single relaxation is given by $TC_{neuro-comp}/R$. Computation of a solution by a Hopfield neural network requires processing time for setup and initialization as represented by $TC_{total} - TC_{neuro-comp}$, and time for one relaxation period which is given by $TC_{neuro-comp}/R$. The sum of these two periods provides the time in terms of the number of timer ticks for computation of a solution as a normalized value. Multiplying this sum with the time interval value then yields the total time for the execution of the Hopfield optimizer application or its time complexity measurement. Accordingly, the normalized time ($T_{normalized}$) is
the product of the two quantities, namely the normalized value for the total number of timer ticks for one complete episode of convergence and the time interval ($t_i$). Therefore, the normalized time can be calculated as:

$$T_{\text{normalized}} = \left( \left[ (T_{\text{total}} - T_{\text{neuro-comp}}) + T_{\text{neuro-comp}} / R \right] \times t_i \right)$$  

(2)

Each mote wakes up from sleep once every time period (which is a controllable simulation parameter) through its dedicated timer. Timers on each mote in the WSN are not synchronized globally: every timer operates independent of others and yet fires with the same time period as other timers on other motes in the sensor network. Accordingly, each neuron output gets updated once per one time period asynchronously of other neuron outputs. The timer code in nesC for each mote is shown in Figure 1. Upon entry into the convergence phase, each mote wakes up per its schedule of its timer and updates the output of its neuron and broadcasts the newly-updated neuron output value to its one-hop neighbors.

```nesC
//Implements the timer and starts the application
command result_t StdControl.start() {
    //Initialize variables for the first time
    InitForUpdate();
    //Start the timer
    return call Timer.start(TIMER_REPEAT, 1000);
}

//Stops the timer and terminates the application
command result_t StdControl.stop() {
    return call Timer.stop();
}
```

Figure 1: nesC code for timer implementation

3.1.4 Simulation study

The simulations were performed for wireless sensor networks with mote counts of 250, 500, 750, 900 and 1,000 on the TOSSIM simulator. The simulations are repeated 10 times with sleep periods (time intervals) of 0.1 second, 0.2 second, 0.5 second, and 1.0 second for each mote count. WSN motes were randomly and uniformly distributed over a two-dimensional square-shaped area with dimensions of 100×100 units squared. There are two built-in radio models in the TOSSIM environment: “empirical” and “Fixed radius”. In the “Fixed radius” model, all motes within a given fixed distance of each other have perfect connectivity, and no connectivity to other motes. The “empirical” radio model is based on an outdoor trace of packet connectivity with the RFM1000™ radios. Simulations were run for radio transmission radius values of “empirical” since this option models the reality with much more fidelity.

Simulation results for the time complexity metric for mote counts of 250 to 1000 are presented in Table 1. Computation time doubles for the mote count of 1000 for varying the time interval from 0.2 second to 0.1 second. It is also notable that the normalized computation time ($T_{\text{normalized}}$) is comparable for all time intervals in the range of 0.2 second to 1.0 second when the mote count is 1000. The computation time of time interval of 0.1 second for 1000 motes is much higher than those for others, which is double the computation time of time interval of 0.2 second and is about three times of the computation time of time intervals of 0.5 second and 1 second. As data indicate, the time cost of simulation decreases with the increase in the time interval. When the time interval value is larger, motes stay awake for longer subperiods of time, which gives more opportunities to access the communication medium, and transmit or receive messages. The delays in medium access and message communications will likely be less for longer time periods, and yet, this is at the expense of increased energy consumption. With the increases in the mote count, the computation time tends to increase and this increase is fundamentally non-linear. As the number of motes increases, given that the deployment area size is fixed for any mote count in the WSN, the number of one-hop neighbors increases. This necessitates any mote to communicate with many more other (one-hop neighbor) motes. As an illustrative example, consider a 10×10 units squared sub area within the 100×100 units squared deployment zone: the number of motes residing within this 10×10 unit squared sub area will increase from approximately 2.5 to 10 for mote counts of 250 and 1000 in the WSN, respectively. There will be much more contention for medium access and many more messages to exchange among the neighbors. Accordingly, this increase in the number of one-hop neighbors is responsible in the increase for the overall simulation time for larger mote counts in the WSN. Therefore, observations related to time measurements emerge as reasonable and meaningful which indicates that the proposed methodology for assessing the time complexity is promising for being relevant and useful.
4. Conclusions

This paper proposed an approach to assess and profile the time complexity of an application executing in the TOSSIM environment. The approach was demonstrated through an artificial neural network, Hopfield recurrent neural network, application embedded across a wireless sensor network on a one-neuron-per-mote basis. The Hopfield neural network was configured as a static optimizer to solve for a solution of a minimum weakly-connected dominating set problem for a graph model of a wireless sensor network topology. Simulation study results established the validity of the time cost measurements through the proposed approach.

References