Cellular Simulation of Asymmetric Energy Requirements in Wireless Sensor Networks

Mohammad El-Shabani
Department of Electrical Engineering and Computer Science
University of Ottawa, Ottawa, ON, Canada
m.elshabani@uottawa.ca

Mohammad Moallemi, Gabriel Wainer
Department of Systems and Computer Engineering
Carleton University
Ottawa, ON, Canada
{moallemi, gwainer}sce.carleton.ca

***Abstract*—*We discuss the importance of careful energy requirement planning in Wireless Sensor Networks (WSN) to optimize and reduce cost. Different cellular discrete-event models of WSN are presented and visualized. An experiment is carried out using the CD++ tool. Different results are presented, showing a range of realistic scenarios. The scenarios show the importance of energy management in these networks.*

Keywords — WSN, Energy, CD++, Cell-DEVS, requirement planning

I. Introduction

A Wireless Sensor Network (WSN) consists of spatially distributed set of nodes that are able to communicate in an ad-hoc manner with no need for infrastructure [1]. Each node includes processing power, multiple types of memory, an RF transceiver, a power source (e.g., batteries and solar cells), and variety of sensors and actuators [2]. Each node functions as a wireless router connected to a set of neighboring nodes with packet routing capability. A routing algorithm tries to optimize the path based on selected parameters, such as the number of hops or energy consumed.

One of the potential applications that drive research in WSN is data collection. This application is envisioned as being able to deploy hundreds or thousands of enabled nodes in random or specific locations, where the nodes are able to auto-configure by setting up ad-hoc connections. For example, nodes could be thrown out of a helicopter in human non-accessible areas such as volcanic centers or rainforests. Other data collection applications include air quality monitoring in cities, water quality monitoring, structural health as in building and bridges, and emergency planning.

The WSN nodes usually have constrained resources to reduce costs, and limited energy consumption. Each node usually has a microcontroller chip, and a lightweight operating system such as TinyOS [3] or ContikiOS [4], which makes creating new applications much easier. A node is usually equipped with certain sensors to collect data about its environment in a specified variable rate.

A sink is used to collect the data gathered by the nodes. All data collected is forwarded to the sink, which acts as a gateway to the WSN; or alternatively, it forwards all the data using some other technology such as GPRS mobile data network or IEEE 802.11 WLAN to be stored in a database. A sink is able to do that because it is usually a more powerful device and is able to handle higher data transmission rates and energy consumption. Such a node has either power connectivity or is equipped with an energy collection method such as a solar panel.

Energy efficiency is one of the most important factors in WSNs, since nodes must run for longest possible period of time while powered by a constrained battery. It is impractical to change the batteries of a big number of nodes, a process that needs to be done for hundreds or thousands of nodes, which could be difficult or inefficient to access in individually. Energy is consumed by a number of activities or states within a node, for example, computation activity in the MCU, data transmission, data reception, idling and sleeping. The most energy consuming activity is data transmission, and the other factors fade in comparison. This is why, it is considered more efficient to do more computation, if it can reduce the size of the data transmitted. Therefore, a node's lifetime is measured by the amount of data it transmits.

The sink connects the WSN to the outside world; thus, data collected is forwarded to a sink either continuously or on-demand. Energy requirements of the nodes around the sink are asymmetrical, since, the closer nodes will be forwarding more traffic towards the sink. In addition, obstacles in the field create additional variance in energy consumption. This will be illustrated in our model and will be discussed later.

The variance of energy requirements of nodes makes it harder to plan the power source capacity. It creates a compromise between the early death of nodes around the source, or wasted battery nodes away from the sink. There are many possible solutions such as deploying larger number of nodes around the sink, which are put to sleep according to a topology control algorithm.

Wireless Sensor Network modeling and simulation is a very active research area due to the complex analysis of the applications envisioned for these networks, as well as the high cost of deployment and maintenance of the sensors. However, the large amount of intensive research dedicated to this area is generated by unique problems that require novel solutions. This is because WSN is different from the other types of conventional networks that have been developing for the last three decades [2]. Simulation tools are commonly used to predict the behavior of WSNs; in fact, they are used for all kinds of distributed and real-time systems, since it is very difficult to prove their behavior deterministically. Besides, it is too expensive and time consuming to test actual implementations.

DEVS is a well-known formal discrete-event M&S methodology, which is based on generic dynamic systems theory. The discrete-event nature of DEVS provides event-driven response to external stimuli, which can represent sensor reaction to the environment and resource limitations. On the other hand, discrete-event methodologies provide continuous timing of the events. In particular, DEVS includes well-defined coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and support for repository reuse.

Cellular state-machines (such as Cellular Automata (CA) [6]) are efficient spatial modeling alternatives for M&S of WSNs. In this branch of discrete dynamic systems, which space is represented by a regular grid, with each cell being a state machine, the time advances in a discrete manner, triggering state changes in the cells, based on the value of their neighboring cells. This theory is used in physics, complexity science, theoretical biology, microstructure modeling, and spatial modeling. The Cell-DEVS formalism [7] is derived from both CA and DEVS. The Cell-DEVS formalism solves the problem of unnecessary processing burden in quiescent cells, and it allows for a more efficient asynchronous execution.

The energy requirement of the nodes is one of the topics receiving the most research attention due to the fact that, it is the most limiting factor when it comes to implementing WSNs. In this paper, energy requirement variance of nodes in a WSN is modeled using the Discrete-Event System Specification (DEVS) formalism [5]. The goal is the use of modeling and simulation (M&S) to test and understand the operation of networks. This is done by emulating certain behavior up to the point where we could say that it matches real behavior within the experimental frame of interest. The use of simulation has become so popular due to the complexity of implementing and proving the correctness of distributed and real-time systems, which makes it very convenient to model and simulate this behavior since it reduces the time, effort and associated costs.

II. Related Work

The energy management issue in WSN is considered as an important requirement in these networks such that low-power hardware components and customized system architectures are required to build them [8]. In WINS project developed in University of California in Los Angeles [9] the specific parameters of WSNs namely higher tolerance to latency and low sampling rates are taken into account to develop low-power hardware. The advance of hardware along with software in these networks has made it very difficult for engineers and programmers to predict the behavior of large WSNs such as *Smart Dust* [10] networks. Therefore, M&S can play a great role in providing a realistic representation of these networks that allows for more efficient assessment of the performance and efficiency of these networks in the field as well as reducing the costs of testing and providing a safe and practical test bed for verification of the sensors in the actual environment [11].

Analytical computational are also investigated in [12] where two network scenarios are studied: one including arbitrarily located nodes and traffic patterns, the other one with randomly located nodes and traffic patterns. Another study presented in [13] provides mathematical computations on solving the problem of optimal data distribution and data collection, and analytically evaluate the time performance of the solutions. However, analytical solutions are mostly focused on network routing and data distribution and do not provide tangible or visual representation of the networks and are limited by many factors.

A number of WSN simulators exist that are mostly concerned with the node management and routing protocols in these networks. J-Sim [14] is a java-based object-oriented WSN simulator that provides scripting language for defining network topologies and routing algorithms like NS-2. It is built on the WSN simulation framework of both the ACA and INET simulation tools [15] and implements several well-known localization, geographic routing, and directed diffusion protocols. These protocols can be readily implemented by extending the object classes defined in the framework and customizing their behaviors.

In [16] a Markov model of a sensor network is presented in which sleep mode in nodes is used to investigate the system performance in terms of energy consumption, network capacity, and data delivery delay. The model is limited to analytical model and does not provide a simulation of the network.

Based on the existing works, most of the efforts are either analytical calculation of network parameters or simulation of the network routing protocols and data transfer. Therefore, our motive here is to provide a tangible and configurable cellular simulation of WSNs to analyze the power management as well as efficiency in these networks by using a discrete-event approach. Our model also provides a visualization of the environment

and field-base conception of the sensors. This will allow us to measure the performance factors and be able to modify network parameters easily to investigate different scenarios.

Cell-DEVS is an extension to DEVS that allows defining cellular models with explicit timing delays. A Cell-DEVS model is a lattice of cells holding state variables and a computing apparatus, which is in charge of updating the cell states according to a local rule. This is done using the current cell state and those of a finite set of nearby cells (called its neighborhood). Cell-DEVS improves execution performance of cellular models by using a discrete-event approach. It also enhances the cell's timing definition by making it more expressive. Each cell is defined as a DEVS atomic model, and it can be later integrated to a coupled model representing the cell space. Cell-DEVS models are informally defined as shown in Figure 1.

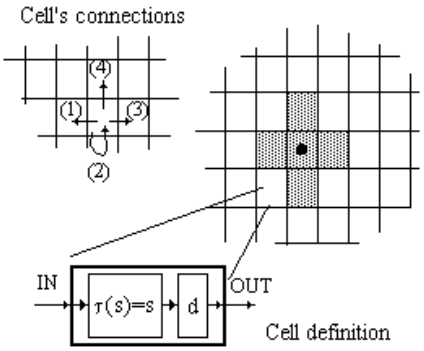

Figure 1. Cell-DEVS model [7].

CD++ [17] is an M&S tool that provides a development environment for implementing DEVS and Cell-DEVS models. DEVS atomic models can be developed and incorporated into a class hierarchy programmed in C++. Coupled models can be defined using a built-in specification language. Cell-DEVS models are built following the formal specifications for DEVS, and a built-in language is provided to describe the behavior rules. The language is based on the formal specifications of Cell-DEVS. The model specification includes the definition of the size and dimension of the cell space, the shape of the neighborhood and borders. The cell's local computing function is defined using a set of rules with the form POSTCONDITION DELAY {PRE-CONDITION}. These indicate that when the PRECONDITION is satisfied, the state of the cell will change to the designated POSTCONDITION, whose computed values will be transmitted to other components after consuming the DELAY. If the precondition is false, the next rule in the list is evaluated until a rule is satisfied or there are no more rules. CD++ was used to simulate and visualize the energy requirement variance in WSNs. An advanced version of CD++ [19] was also used to provide this functionality, which enabled the improvement of the original model and allowed the execution of more realistic test cases.

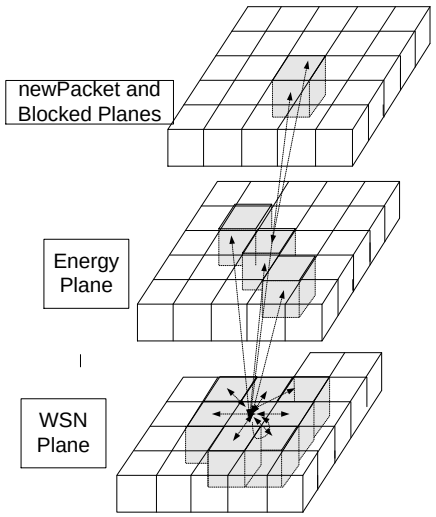

Figure 2: Cell neighborhood Cell-DEVS model implemented in CD++

The model uses three planes, each containing a different behavior.. The first plane is used to control the number of packets to be forwarded in the next step (packets in buffer). The second plane is monitoring energy consumption. The third plane is used to generate new packets. A multi-plane neighborhood definition connects the cells of all planes together to easily access data from the associated cells in each plane. Figure 2 illustrates this design.

The first plane is a rotation plane in which all nodes are initialized to a random number, and they rotate from 0 to 9. The other plane is set to 1 when the corresponding node in the rotation plane is equal to 1. Thus, all nodes generate packet with the same period; however, they are not synchronized.

The sink is located in the lower left end in the suggested model, because it is simpler to follow the shortest path in such a scenario correctly. To be able to locate a sink, we would require additional planes, which will increase the computation time of the simulation (as the number of cells would increase).

In addition, nodes in the *main* plane read the states of *energy* and *blocked* planes included in the neighboring set. This is to make the decision on where to forward packets. Cells in the *energy* plane set themselves to a very large number, if the associated cell in the *blocked* plane is set. Therefore, other nodes will not assume that packets are forwarded by the *blocked* cells.

The three extra planes simplify and reduce the rule-set because they help account for many special cases in an ingenious way. For example, the *new packet* plane reduced the set of rules to half. In addition, before adding the *blocked* plane, a *blocked* cell was represented by setting a cell in the main plane to -1. However, the model required more than 20 rules to account for special cases.

After introducing the plane, the cell in the main plane remains zero. This, in addition to setting corresponding cells in the energy plane to a very large number, eliminated all but one special case because they fit correctly.

III. Models Definition

The model defined involves a single plane of cells and each cell has 8 state variables. A size of 20x20 cells was chosen. To simplify implementation, a modular design is used where forwarding is separated from routing to simplify the implementation.

An initialization phase starts by setting up the routing parameters or nodes. Each node that does not have routing information looks if one of its neighbors sends the rout information to it or if one of them is a sink. Then, it defines the route by setting a route variable pointing at that node. There are actually three routing variables: one is used for forwarding, and two others, which are alternative options. The main routing variable is evaluated at each time-step according to the energy consumption of possible destinations. It chooses the one that has consumed less energy. The first alternative is the first route found by the node as described earlier, while the second alternative is any other possible path that is not the same as the first one and where the other node does not point back to the original one.

After the initialization phase is done for some cell, new packets start being generated one for every 10-time steps. At the beginning, a rotation is set up and for each 10 time steps, all the cells send a single packet at a random point in that period. This is why a rotation variable is required.

At each time-step, each node checks the routing of neighboring cells and collects packets that are pointing towards it by looking at the neighbors' packet variable in addition to packets generated by it. Then, it sets that number to its own packet variable.

The energy consumed by each node is evaluated to be the total number of packets a node forwarded; thus, the energy consumed variable counts that number and this variable is used to visualize the results of the model.

An additional *entity variable* is required to be initialized at the beginning to distinguish between regular nodes, empty cells, and blocking cells. This variable is used to create various test cases that will be shown later.

The model was implemented in CD++ defined as in Table 1. The rules are organized in sets to facilitate explanation:

Set 1: sets route for exceptional cases according to entity, if blocked, set to -1 and if sink set to 1. These are preset constants not required for operation, but are used for visualization.

Set 2: if routing is not set yet, and is set at one of the neighbors, set routing to that neighbor. This is the initial phase that establishes shortest path reachability.

Set 3: This phase establishes the alternative paths. route2 is set to the next neighbor that is reachable to the sink. It makes sure that none of the routing alternatives of that neighbor direct traffic at the original cell.

Set 4: In the rare occasion of two neighboring cells directing traffic at each other, the following set of rules takes care of fixing that.

Set 5: The following rules take care of choosing a route from the two alternatives. It attempts to create some sort of load balancing be forwarding the neighbor with the lower total energy consumption.

Set 6: This rule is the last resort; it is executed if no other previous rule is.

```
type : cell                    dim : (20,20)
delay : transport            border : unwrapped
neighbors : wsn(-1,-1) wsn(-1,0) wsn(-1,1)
neighbors : wsn(0,-1)  wsn(0,0)  wsn(0,1)
neighbors : wsn(1,-1)  wsn(1,0)  wsn(1,1)
localtransition : wsnrule
statevariables : route route1 route2 entity energy
packets rotation newPacket
neighborports : route1 route2 route entity energy
packets rotation newPacket
initialvariablesvalue : wsn.stvalues

[wsnrule]
#set 1
rule : {~route := $route1; ~route1 := $route1;
~route2:= $route2;#macro(routinePort)}{$route1 := -1;
$route2 := -1;#macro(routineAssign) } 10 {$entity = 1}
rule : {~route := $route1; ~route1 := $route1;
~route2:= $route2;#macro(routinePort)}{$route1 := 1;
$route2 := 1;#macro(routineAssign) } 10 {$entity =2}

...

#set 6
rule             :             {~rotation             :=
$rotation;#macro(routinePort)}{ #macro(routineAssign) }
10 { t}
```

Table 1: Model implemented in the enhanced CD++.

The model uses a set of rules defined as macros as follows::

```
#BeginMacro (routineAssign)
$rotation := remainder ($rotation+1,10);
$newPacket :=  ifu($rotation = 1 and $route !=
0 ,1,0,0);
$packets := if ( $entity != 1, $newPacket
+ ifu( (0,1)~route = 4 , (0,1)~packets ,0,0)
+ ifu( (-1,0)~route = 5 , (-1,0)~packets ,0,0)
+ ifu( (0,-1)~route = 2 , (0,-1)~packets ,0,0)
+ ifu( (1,0)~route = 3, (1,0)~packets ,0,0),-1);
$energy := if ( $entity != 1 , $energy +
$packets , 10000 );
#EndMacro

#BeginMacro (routinePort)
~rotation := $rotation;
~newPacket := $newPacket;
~packets := $packets;
~energy := $energy;
#EndMacro
```

Table 2: The macros

The "routineAssign" macro takes care of assigning the correct values to a set of variables; this macro is executed every time since it is included with all the rules. The rotation variable is incremented at every time step, and is initialized to random number between 0 and 9 at initialization.

The packets variable indicates the number of packets queued at the cell in that time step; it is calculated at each time step by collecting the packets destined to it from the neighboring cells. The energy variable indicates the total number of packets forwarded.

IV. Simulation Results

Various testing scenarios were considered to cover best cases and realistic situations. The rest of this section shows the results produced by these scenarios in the following figures. Note that the most interesting results produced for each scenario are shown here. The following were the chosen results:

1- The number of packets queued at each node at the chosen time step, where a darker red color indicates a larger number of packets.

2- Total energy consumption at each node at that time step, where a darker red color indicates higher total energy consumption.

3- The routing map at that time step, where each color refers to forwarding in a certain direction: red refers to the right, green to the left, yellow to upwards and pink to downwards. The blue color indicates a sink.

In Scenario 1, no blocked cells are considered and all nodes are allowed to forward packets naturally. In addition, only one sink is set at the lower left corner similar to the old model. This should provide a uniform fade of the level of energy consumption from the lower left corner (the sink) to the upper right corner.

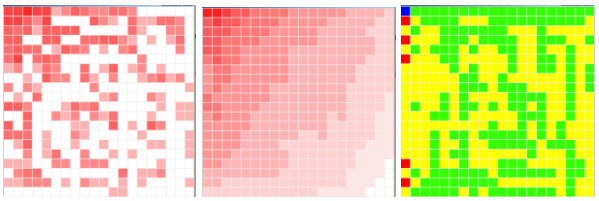

Figure 3: a) Number of packets queued at each node on time step 80 b) Energy consumption c) Routing map at time step 80 for the basic case

The results shown in Scenario 1 are as expected; the routing map is dynamic and the one shown is a snapshot. There is a higher packet density towards the upper left cells (which represent the WSN nodes) in Figure 3a. This explains the higher energy consumptions at the higher left cells than the lower right in Figure 3b, as the latter forwarded fewer packets. The routing map shows that most of the cells are forwarding upwards or to the left, and this makes sense. The few cells that behave differently forward to the right in order to distribute energy consumption. This is done this way because they detect higher total consumption at the upper cell than the one to the right.

Another two scenarios (scenario 2 and 3, shown in the following figures) use multiple sinks. The first includes two sinks at the upper left and right corners and another midway of the lower row, while the other has a sink at each corner of the model.

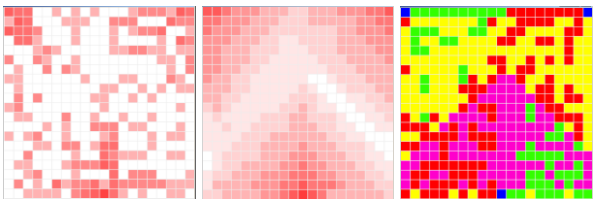

Figure 4: a) Packets at time step 80 for the multiple sinks case; b) Energy consumption and c) Routing map for Scenario 2

Figure 5: Packets queued at each node at time step 80; energy consumption; and c) Routing map for Scenario 3.

These test cases (Scenario 2 and 3) show a much better distribution of packets and energy consumption. Figure 5a shows that the density of packets is higher towards the corners, which means that those cells forward more packets. This is also conveyed by the energy consumption distribution in Figure 5b. Figure 5c shows the four sinks in the four corners (the blue cells); in addition, it shows the forwarding directions at that instance.

Our following test (scenario 4) included blocking obstacles to the scenario 2.

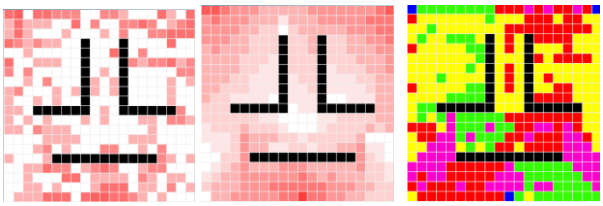

Figure 6: a) Number of packets queued at each node at time step 80; b) Energy consumption; and c) Routing map with obstacles.

The black cells in the three figures above represent dead regions; they could either represent a dead node, a barrier preventing communication or just an empty space with no nodes deployed; in any case, no forwarding happens through these cells.

This simulation result shows an interesting routing map established, and the longer paths taken by packets.

In addition, in scenario 5, one sink is centered at the middle of the cell space. This shows how placing the node in the middle changes the variance of energy consumption.

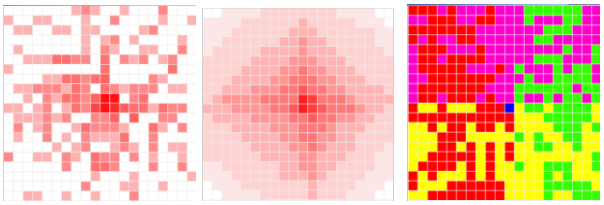

Figure 7: Number of packets queued at each node at time step 80; b) Energy consumption; c) Routing map.

Figure 7b shows an energy consumption distribution similar to scenario 1.

Finally, we show a bad blocking case (scenario 6) in which the sink has been placed in the middle, and an almost closed circle of blocking cells is around it.

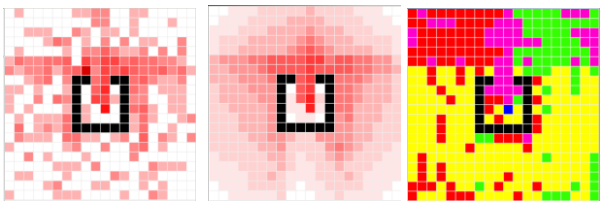

Figure 4: The number of packets queued at each node at time step 80; b) Energy consumption; c) Routing map.

Finally, we include the simulation results of two realistic cases (scenarios 7 and 8). In the first example, there are four sinks placed strategically in the cells (5,5), (5,15), (15,5), (15,15). This shows how such a WSN implementation can reduce variance.

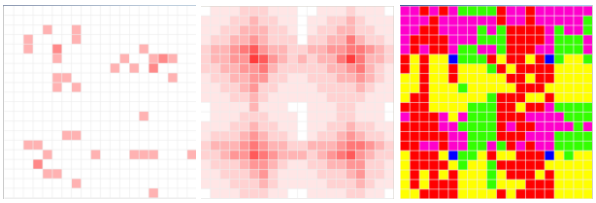

Figure 5: a) Number of packets; b) Energy consumption; c) Routing map for a better location scenario for sinks.

Figure 9a shows how a better choice for the location of the sinks results on a smaller number of packets in the network.

In addition, a realistic blocking case was simulated in a field with almost 25% randomly placed blocked cells. This case shows how packets are forwarded around the blocked nodes; and thus, further energy consumption variance can be seen in those areas.

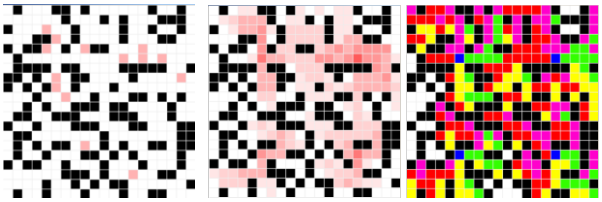

Figure 10: a) Number of packets; b) Energy consumption; c) Routing map for a randomly distributed set of obstacles.

In a realistic scenario, a certain ratio of dead spots should be expected. In addition, as shown in the routing map in figure 10c, some cells that do contain active nodes are not active (the white cells); this is because these cells are surrounded with dead cells.

After implementing an improved version of the model using a new version CD++, and collected some basic performance metrics in order to evaluate the performance the different models.

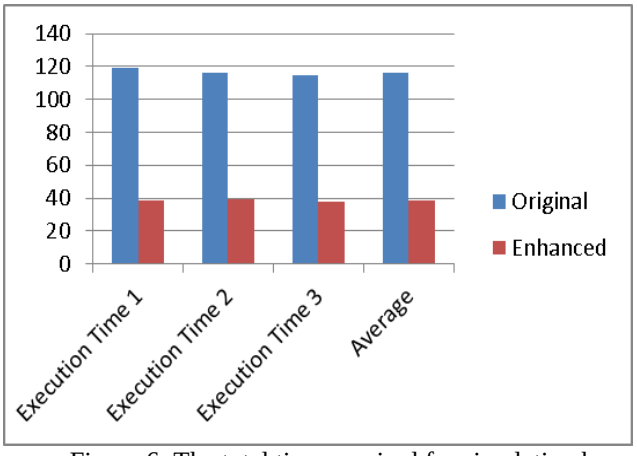

Figure 6: The total time required for simulation by the CD++ tool (comparison between original and enhanced tools)

As illustrated in the figures above, the total number of messages actually increased using the enhanced version of CD++; however, the execution time required to simulate the model was reduced significantly to almost 30% of the original time required.

V. Conclusion

In this paper, we modeled WSNs using the enhanced version of Cell-DEVS in which multiple values are allowed in each cell. In the experiments provided, the variance of energy requirement of WSN nodes was illustrated and various deployment cases were tested as planned. It was showed that careful planning of energy requirements is needed to reduce and optimize costs.

In addition, the possibilities provided by the enhanced version of CD++ are demonstrated. It was shown how scenarios can be built and tested rapidly due to the tool's unique architecture. In addition, the enhanced version provided significant CPU time improvement.

VI. References

[1] W. Dargie, and C. Poellabauer, "Fundamentals of wireless sensor networks: theory and practice", John Wiley and Sons, 2010 ISBN 978-0-470-99765-9, pp. 168–183, 191–192.

[2] J. A. Stankovic, "Wireless sensor networks." computer 41.10 (2008): 92-95.

[3] TinyOS official website available at: http://www.tinyos.net/, accessed Aprill 2013.

[4] ContikiOS official website available at http://www.contiki-os.org/, accessed April 2013.

[5] Zeigler, B., T. Kim, and H. Praehofer, "Theory of Modeling and Simulation", Academic Press, ISBN: 0127784551, 2000.

[6] Wolfram, S. (1986). Theory and applications of cellular automata.

[7] Wainer, G. A. (2009). Discrete-event modeling and simulation: a practitioner's approach. CRC.

[8] W. Rabiner Heinzelman, A. Chandrakasan, H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," Proceedings of the 33rd International Conference on System Sciences, January 2000.

[9] G. Asada, M. Dong, T.S. Lin, F. Newberg, G. Pottie, W.J. Kaiser, H.O. Marcy, "Wireless integrated network sensors: Low power systems on a chip," Proceedings of the 24th IEEE European Solid State Circuits Conference, 1998.

[10] Smart Dust project website available at: http://robotics.eecs.berkeley.edu/~pister/SmartDust. Accessed April 2013.

[11] Liu, Jason, et al. "Simulation modeling of large-scale ad-hoc sensor networks." European Simulation Interoperability Workshop. Vol. 200. No. 1. 2001.

[12] P. Gupta and P. R.Kumar, "The Capacity of Wireless Networks," IEEE Trans. on Information Theory, vol. 46, Mar. 2000.

[13] C. Florens and R.McEliece, "Packet Distribution Algorithms for Sensor Networks," IEEE Infocom, San Francisco, CA, Mar. 2003.

[14] Sobeih, Ahmed, Jennifer C. Hou, Lu-Chuan Kung, Ning Li, Honghai Zhang, Wei-Peng Chen, Hung-Ying Tyan, and Hyuk Lim. "J-Sim: a simulation and emulation environment for wireless sensor networks." Wireless Communications, IEEE 13, no. 4 (2006): 104-119.

[15] H.-Y. Tyan and J. C. Hou, "JavaSim: A Component-Based Compositional Network Simulation Environment," Proc. Western Simulation Multiconference — Communication Networks and Distributed Systems Modeling and Simulation (CNDS'01), Jan. 2001.

[16] Chiasserini, C-F., and Michele Garetto. "Modeling the performance of wireless sensor networks." INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies. Vol. 1. IEEE, 2004.

[17] G. Wainer: "CD++: a Toolkit to Define Discrete Event Models", Software, Practice and Experience, Wiley, Vol. 32, No 3. pp. 1261-1306. November 2002.

[18] Blerim Qela, Gabriel A. Wainer, Houssein Mouftah, "Simulation of Large Wireless Sensor Networks using Cell-DEVS", Proceedings of the Winter Simulation Conference, Austin, TX, 2009.

[19] A. López, G. Wainer. Improved Cell-DEVS model definition in CD++. P.M.A. Sloot, B. Chopard, and A.G. Hoekstra (Eds.): ACRI 2004, LNCS 3305.Springer-Verlag. 2004.

[20] E. Glinsky; G. Wainer: "Performance Analysis of Real-Time DEVS models", In Proceedings of Winter Simulation Conference, San Diego, U.S.A, 2002.