Lifetime Analysis of a Sensor Network with Hybrid Automata Modelling

Sinem Coleri*
Dept. of EECS
Univ. of California at Berkeley
Berkeley, CA 94720
csinem@eecs.berkeley.edu

Mustafa Ergen†
Dept. of EECS
Univ. of California at Berkeley
Berkeley CA 94720
ergen@eecs.berkeley.edu

T. John Koo‡
Dept. EECS
Univ. of California at Berkeley
Berkeley CA 94720
koo@eecs.berkeley.edu

ABSTRACT

In this paper, we focus on TinyOS, an event-based operating system for networked sensor motes. We show how to model TinyOS as a hybrid automata with HyTech and verify the correct operation of the system by using safety verification feature of HyTech. Since lifetime is an important metric for sensor nodes that are planned to be deployed once and unattended for long periods of time without maintenance, we perform power analysis of a sensor node by using trace generation feature of HyTech. Furthermore, we simulate a tree sensor network of TinyOS motes by using the programming language SHIFT to determine the lifetime of the network as a function of the distance from the central data collector.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Reliability, availability, and serviceability]; D.2.4 [**Software Engineering**]: Software/Program Verification—*Model Checking*

General Terms

Performance, Verification

Keywords

TinyOS, Sensor Networks, Power Consumption, HyTech, SHIFT, Hybrid Automata

1. INTRODUCTION

Wireless sensor nodes have emerged as a result of the recent advances in low-power digital and analog circuitry, low-power RF design and sensor technology. These inexpensive nodes are designed to operate in a network of hundreds or thousands of sensors to achieve high quality data. The practical application of such systems range from security monitoring in detecting intrusions to traffic monitoring in high-ways.

Figure 1: Photograph for a Representative Sensor Platform

Achieving the goal of cooperative sensor devices requires the design of a software platform to manage and operate the device in the most efficient way. TinyOS [3], a tiny event-based operating system, is developed to address the application-specific characteristics of sensor networks such as small physical size, low-power consumption, robustness and modularity.

One essential requirement for sensor networks is the reliability of applications since sensor devices are planned to be deployed into the area once and unattended for long periods of time without maintenance. However, concurrent interactions between TinyOS components can make the behavior of applications very hard to predict for any input configuration. For instance, TinyOS applications may stop working after some time due to some condition that never occurred while you were testing your application. We show how to guarantee the correct operation of the system by using the verification feature of HyTech [4], which is a symbolic model checker for linear hybrid automata.

Another essential feature of sensor networks is the longest possible lifetime from a specific hardware configuration. The data being sensed by the nodes in a sensor network must be transmitted to a control center or base station so that the end-user can access this data. To conserve power, it is essential that the data is being relayed multiple times towards destination than to increase the transmission strength so that it is received directly by the center. The problem in the multi-hop case is that the nodes closer to the con-

*Sinem Coleri is a Ph.D. student in UC Berkeley

†Mustafa Ergen is a Ph.D. student in UC Berkeley

‡T. John Koo is a Visiting Faculty in UC Berkeley

WSNA'02, September 28, 2002, Atlanta, Georgia, USA.

trol center will relay a large number of packets compared to the nodes away from the station. We perform the power analysis of one node as a function of the distance from the base station by generating a trace from an initial region to a final region by performing forward reachability analysis in HyTech. Moreover, we analyze the overall sensor network performance by grouping the nodes according to their distance from the base station by using SHIFT [8], which is a tool for the description of dynamic networks of hybrid automata.

Section 2 introduces TinyOS and gives TinyOS power characteristics. Section 3 gives the verification and the power analysis of TinyOS through modelling the system in HyTech. Section 4 explains the power consumption over the sensor network with simulation in SHIFT. Section 5 and 6 presents the previous research and future directions respectively. Section 7 concludes the paper.

2. TINYOS

2.1 TinyOS Description

TinyOS is an event-based operating system designed for power-efficient, concurrency-intensive operation and modularity. Since network sensor devices carry only the hardware needed for a specific application, it is important to easily assemble the software components corresponding to hardware components due to memory limitation of these devices. This unusual degree of software modularity without heavyweight interfaces is provided through component-based model of TinyOS.

The components in TinyOS can be classified into three groups. The first group consist of components that are thin abstractions over hardware such as clock, bit level radio components. Second class of components act as a replacement for unavailable hardware such as a byte level radio component on top of a bit level radio component. The third level of components are high-level software components performing control part of the application.

A complete TinyOS application consists of a graph of components and a two-level scheduler. A component has four interrelated parts: a set of command handlers, a set of event handlers, a frame and a bundle of tasks. To facilitate the modularity, each component declares the commands it uses and the events it signals in separate *.comp* file, which are the linked for the complete application in a *.desc* file. The actual functionality of the components containing a frame along with the code with command handler, event handler and task executions is specified in a *.c* file. A frame is a statically allocated part of the memory and contains the state of the component.

Commands and events are just function calls across the components that provides a feedback to the caller through a success/failure status. Commands are non-blocking requests to lower-level components. Lower-level components have the handlers corresponding to each command coming from upper level components. On the other hand, events are invoked to deal with hardware events either directly or indirectly. The lowest level components have handler connected to hardware interrupts. Inside this handler, a small amount of work on the component's state is performed and another event is generated. A fountain of processing occurs within the context of components' state while going upward through events and downward through commands.

Instruction Type	Energy per cycle (nJ)	Energy per instruction (nJ)
idle	0.001	0.001
arithmetic/ logic	3.41	3.41
memory read	3.66	7.32
memory write	3.75	7.50

Table 1: Energy Consumption of instructions in Smart Dust prototype

Device	Energy per quantum
Photo	0.08-0.28 nJ/cycle
ADC	4.62-3.95 nJ/conversion
RFM send	1 μJ
RFM receive	0.5 μJ

Table 2: Energy Consumption of external modules in Smart Dust prototype

Tasks provide a way to incorporate arbitrary computation into event-driven model. They are atomic with respect to other tasks but preempted by events.

TinyOS currently has a two-level scheduler. The first level contains events and commands. Small amount of work associated with hardware interrupt upon components states are performed immediately. The second level contains tasks. When a task is posted inside a command or event handler, it is posted into a FIFO queue. The tasks are executed when CPU has no events or commands to run and can be interrupted by a hardware event. When there is no task, event or command to execute, the processor is put to sleep while leaving the peripherals operating so that the system can wake up with any hardware interrupt.

2.2 TinyOS Power Characteristics

A networked sensor has a predetermined operation in contrast to a general purpose computer, which includes sensing, slight processing and communicating the results to a central data collector. The determination of the power consumed in common operations can help the designers to determine the average power required for their application.

The study of power usage is performed for the SmartDust prototype, which includes an Atmel 90LS8535 processor externally clocked at 4MHz, a co-processor unit, an RF Monolithics 916.50 MHz transceiver and an analog light sensor, in the paper named Power and Control for Networked Sensor by E. J. Riedy and R. Szewczyk [5]. Table 1, 2, 3 summarize the findings obtained from a digital oscilloscope triggered by a set of microbenchmarks measuring various primitive operations.

Operation	cost (cycles)
Post an event	10
Post a command	10
Post a thread to scheduler	46
Interrupt (software overhead)	60

Table 3: Cost of the Basic Operations and Overheads in TinyOS

3. VERIFICATION AND POWER ANALYSIS OF TINYOS THROUGH MODELLING AS A HYBRID AUTOMATA

3.1 Modelling of TinyOS as a Hybrid Automata

Hybrid Automaton is a mathematical model for a class of dynamical systems that exhibit both discrete and continuous behavior. The evolution of discrete states is governed by a finite automaton while the evolution of continuous states is specified by differential equations. Linear hybrid automaton is a class of hybrid automata that the flow equations of the continuous states are linear in time. Since the flow equations, invariant conditions, initial conditions and jump conditions are convex linear predicates and the flow condition is a predicate over the derivative of variables only, there exist efficient algorithms for computing reachable set, such as the function *Post* on state assertions.

Linear hybrid automata is very suitable for modelling TinyOS operation due to one-to-one correspondence between their functionalities. The component-based structure of TinyOS facilitates its description in terms of automata. Each TinyOS component corresponds to an automaton.

TinyOS components communicates with each other through events from lower level to higher level components and commands from higher level to lower components. These events and commands correspond to synchronization events in hybrid automata. However, the synchronization events are instantaneous while the TinyOS events require a specific number of clock cycles as given in Table 3. This effect is accounted by staying in a dummy state for this number of clock cycles after making the transition with event. In addition, events and commands in TinyOS are just across component function calls so they have a returning value showing the end of the function and the success/failure of the processing. We have modelled the return of each event as another event. When the command or event processing is finished, the completion event is sent to the component that has generated this event. When the returning value of the function is used, two types of completion events are used: positive completion event and negative completion event.

TinyOS clock cycle corresponds to a hybrid automata discrete step in the model of TinyOS while the energy consumption is kept in a continuous variable *energy*. This *energy* variable is increased abruptly in sensing, receiving a bit and transmitting a bit while it is increasing with different rates in idle and active modes of the processor continuously. The overall view of the TinyOS hybrid automata model is shown in Figure 2. In addition to modelling TinyOS components, we have also added a packet generation automaton to simulate the behavior of the environment. The packets are injected to the system by *packet_generation* and *application* components. *Packet_generation* component represents the generation of packets received from the neighbors. It generates a periodic *rfm_clock* event, which represent the periodic radio clock interrupt in TinyOS hardware platform. Then the packet generation is done by setting a variable *bit* to 0 or 1 according to the state of the *packet_generation* component. Application component represents the generation of packets to be transmitted either as a result of a periodic sensing operation or the forwarding of the received packets.

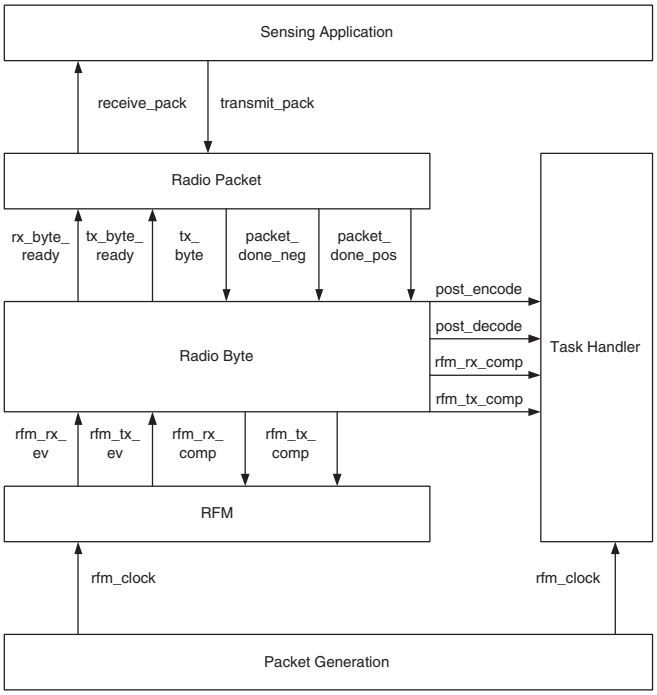

Figure 2: Overall view of the TinyOS Hybrid Automata Model

rfm_clock event generates *rfm_rx_ev* event if RFM component is in receive state and *rfm_tx_ev* if RFM is in transmit state. If *rfm_rx_ev* is generated, radio byte component can either execute some instructions and send *rfm_rx_comp* or send *rx_byte_ready* to upper level to notify that one more byte is received. Upon reception of *rx_byte_ready* event, radio packet component either notifies the end of event execution by *packet_done_neg* or sends the complete packet to the application with *receive_pack* event and sends *packet_done_pos* as a completion event to show the end of the packet reception. The transmission is done in the same way except that it starts with the *transmit_pack* event generated by the application and *rfm_clock* is used to transmit the bits.

Another important point in TinyOS modelling is task handler. Here, we have used the decoding and encoding tasks to illustrate the modelling of the tasks. The model can be extended further. When the byte component has to decode the bits corresponding to one byte of information or to encode one byte to generate the bits to be transmitted, it just posts these calculations as a task to the task handler due to their time-consuming operation. Task handler component, as can be seen in Figure 2, also has input events *rfm_clock* to be interrupted upon hardware interrupts and *rfm_rx_comp*

and *rfm_tx_comp* to continue execution when there are no event handler to execute.

Each automaton corresponding to a component in TinyOS contains three states for each state in TinyOS model. One of them is the actual state where components wait for an event from lower component. The other state is the energy state, where the automaton stays for the number of clock cycles to execute the corresponding instructions in the event handler plus the number of clock cycles to post the event. The third type of state is the wait state where the automaton waits for the complete event to be occurred when the event handler returns.

Component RFM is shown in Figure 3 as an example. The component initially starts in receive state and stays in receive state when either it is receiving a packet or it is listening for potential packets although there is no incoming packet. The component passes to transmit state as a result of the *transmit_pack* event and stays there until the packet transmission is complete. When the component receives *rfm_clock* event from *packet_generation* component in receive state, it changes its state to *rec_energy* and increases the variable *energy* by *crec*, which is a constant showing the energy consumed in reception of one bit from Table 2. The component stays in *rec_energy* state for *crec_handler* (*crec_handler* is the constant showing the number of clock cycles necessary to execute the instruction in RFM component plus the number of clock cycles used for interrupt handling) time steps. Then it changes its state to *rec_wait* where it waits for the completion event showing the function return. The kind of function return shows the next state to move, receive state upon *rfm_rx_comp* and transmit state upon *rfm_tx_comp*. Similar operations are performed when RFM component is in transmit state at the time of *rfm_clock* event generation.

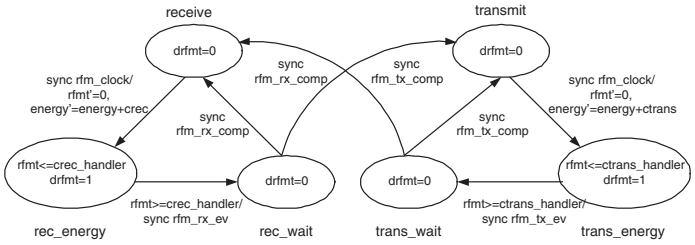

Figure 3: Hybrid Automata Model of RFM Component

A key component in TinyOS modelling is *task_handler*, which determines energy dissipation rate of the system and models the two-level scheduler. *Task_handler* component is initially in idle state. Whenever there is nothing to execute, *task_handler* is in idle state, which shows that the processor is in idle state and consumes energy with a smaller rate. This is specified to be the constant cinactive ($1pJ$ from Table 1). If no task is posted, it moves to *exec* state just to increase the energy rate to cactive($3.50nJ$ on average from Table 1). Whenever a task is posted, the posting of task is performed in state *op_exec*, which takes *cpost_task* clock cycles. Then *task_handler* waits in state *op_wait* until the event execution is completed, which is signaled with either *rfm_rx_comp* or *rfm_tx_comp*. While executing the task, the task can be interrupted by *rfm_clock* event, which puts *task_handler* into *op_wait* state. Then *task_handler* changes its state back and forth between *op_wait* and *op_exec* until the execution of task is complete.

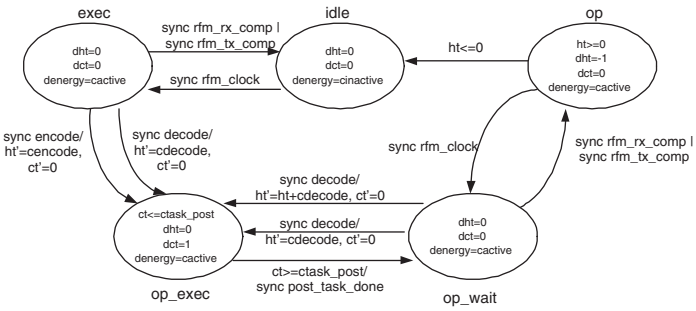

Figure 4: Hybrid Automata Model of Task Handler

3.2 Verification and Power Analysis of a Sensor Node

The reliability of sensor network applications is essential since it may be either impossible or not feasible to replace the sensor nodes. However, concurrent interactions between TinyOS components can make the behavior of applications very hard to predict for any input configuration. The correct operation of sensor networks must be guaranteed under any circumstance.

TinyOS event-driven system is prone to programming errors due to the lack of direct interaction between components although they are directly related. The generation of an event at the wrong place of the program may cause the system operate worse than ideal and may not even be recognized by the developer or may not be experienced by some chance when running the program. An example of this kind of bugs is shown in Figure 5. In TinyOS, when the RFM component receives a bit, it gives this bit to the *radio byte* component. *Radio byte* component does not send any event to radio *packet* component unless it has received one exact byte. Therefore, at the beginning of the packet, until the whole byte is received, the packet component is still in idle state. If application component tries to send a packet at that time and packet component does not check with byte component first as shown in Figure 5, it may send an event to application showing that it has sent packet successfully. However, when it tries to send it to byte component afterwards, it is rejected since byte level is in receive state although the application is positively acknowledged for this packet.

We have performed safety check in HyTech to verify that such conditions cannot be reached. HyTech [4] is a tool for the automated analysis of embedded systems. HyTech can ensure the correct operation of the systems by performing safety and timing verification of the systems, which can be posed in a natural way as a reachability problem. In addition to checking system requirements, this tool can also be used for the generation of a trace from an initial region to a final region by using forward reachability analysis. We can verify the whole TinyOS system by setting the conditions that should not be reached in any way as a final region and performing reachability analysis. For instance, the case in Figure 5 could have been detected by setting the final condition as "radio packet is in transmit and radio byte is in

receive state".

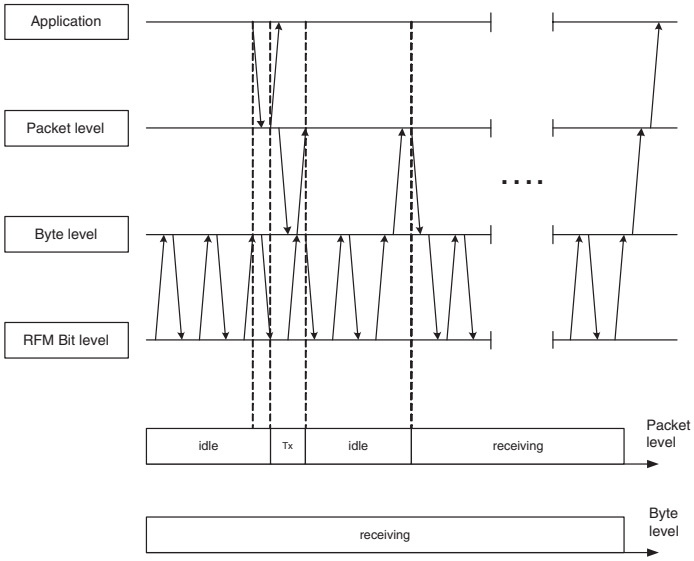

Figure 5: Motivation for the Verification of TinyOS

We have also performed the power analysis of a sensor node by generating a trace of the system with final region as time is greater than the duration of the simulation time. Then the energy variable that is mentioned in Section 3.1 gives us the energy spent during this time. The energy variable increases with rate *cinactive* when the processor is idle and *cactive* when the processor is active. The variable also increases abruptly upon receiving or transmitting a bit or sensing.

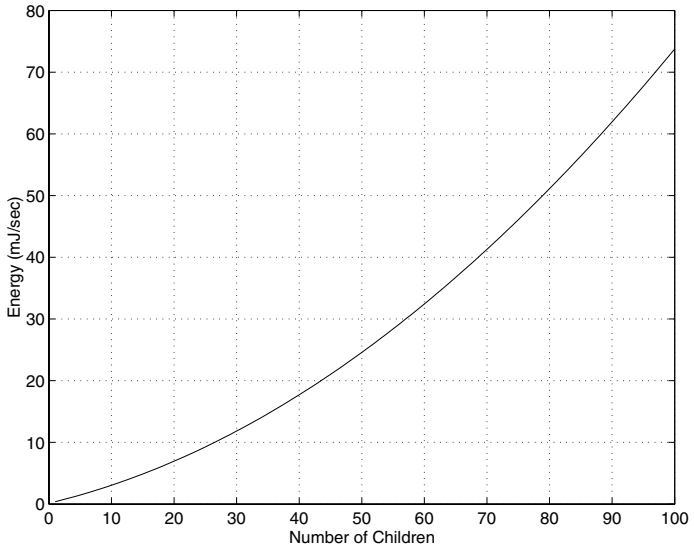

Figure 6: Energy Spent in One Sensor Node with Different Traffic Load

The power analysis is performed for the nodes at different distances from the central data collector. The parameter changed in the simulation is the packet generation rate in the packet generation component to reflect the fact that more number of packets are forwarded as the node is closer to the base station. As can be seen in Figure 6, the energy spent increases with an increasing slope as the number of children of the node increases. The reason is that as the number of incoming packets increases, the node spends most of the time in backoff state trying to transmit the packet in addition to receiving and transmitting packets the number of which is proportional to the number of children.

4. POWER ANALYSIS OF SENSOR NETWORK

A typical application in a sensor web is gathering of sensed data at a distant base station (BS). The sensor nodes are homogeneous and energy constrained with uniform energy. The data from all the nodes need to be collected and transmitted to BS. A simple approach to accomplish this task is for each node to transmit its data directly to the BS. However, if the sensor network is deployed in a big region, this one hop transmission is costly and nodes die very quickly. Another approach is to use multi-hop forwarding. In the common multi-hop configuration, sensor nodes forms a routing tree where the root of the tree is BS. Each node forwards data of its own and its children to its parent in the tree. Another approach is hierarchical multi-hop configuration, where sensor nodes are clustered. In each cluster, one of the nodes acts as the head of the cluster. Then every node in the cluster sends its data to cluster head, which forwards them to the base station. Power analysis of clustered configuration is analyzed in [6], [7].

We focus on multi-hop configuration since this kind of configuration is inevitable when the sensor nodes are deployed in big area. Even if the nodes form a hierarchical network, cluster heads may not be capable of reaching the base station in one hop. As a result, a multi-hop configuration is needed on top of hierarchical configuration.

Power analysis of a sensor network is necessary since network is desired to be in operation state as long as possible. "Operation state" is defined as the state in which the network graph remains connected including the base station with the assumption that route configuration is dynamic and can be reactivated. This performance figure determines the lifetime of a sensor network. Lifetime of a sensor network is desired to be predictable since when the graph become unconnected (i.e there is no path to the base station.), sensor network need maintenance.

We analyze a multi-hop tree-based network in order to observe the power dissipation in regions at different distances from BS. As expected, power dissipation is lower for the nodes apart from the base station since they only forward their own data and higher for the nodes near to the base station since they have more children. Our aim in this analysis is to deploy the network in a clever way with a clear understanding of the relation of the traffic load and power consumption of the sensor nodes. One clever configuration may be to use nodes with energies increasing gradually as we go away from BS. For this to be implementable, regions can be grouped (See Figure 7) and each group contains nodes with different energy.

We use SHIFT [8] as the simulation tool. SHIFT is a programming language for describing dynamic networks of hybrid automata. It consists of components which can be created, interconnected and destroyed as the system evolves. Components exhibit hybrid behavior, consisting of continuous-

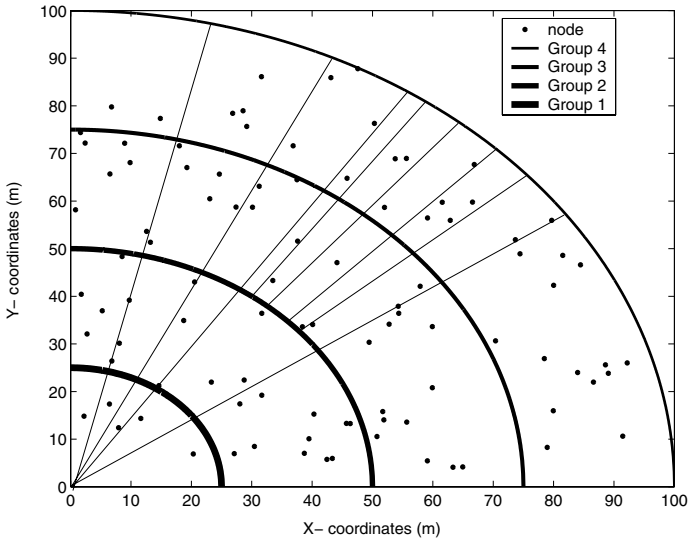

Figure 7: Random 100-node topology

time phases separated by discrete event transitions. Components may evolve independently, or they may interact through their inputs, outputs and exported events.

In a uniform distributed network, we clustered the network into four groups. Power dissipation of a node depends on the number of its children. Children of a node are determined by the geographical distribution. In each group, the nodes are separated into regions with a straight line in which there is only one node and all the lower group nodes occupied between the lines is assigned as children to that node. In each group, energy dissipation of nodes are determined according to Figure 6 and group determination is position based as seen in Figure 7. The modelling of a node is as follows: when a node dies, load of that node is distributed to the nodes in its group. And the total load of the group decreases if there is a death in the lower layer. Hybrid modelling of a node is shown in Figure 8. Each node (n_i^j) has consumed energy X_i^j and power dissipation rate f_i^j, obtained from Figure 6, where $i \in (1,4)$ represents the group number and $j \in (1, N_i)$ represents the ID of the node in the group that has N_i members. Each group is represented with set S_i to signal all the nodes in the group and set $R_i^j = S_i - \{n_i^j\}$ for $j \in (1, N_i)$ to signal a death event to the nodes except itself. When a node dies, it sends *death* signal and the load of the dead node is distributed to the rates of nodes in set R_i^j and the corresponding load that affect the upper groups is decreased from the rates of nodes in S_i of those groups. The distribution of the rate is uniform, more complicated schemes can be done by distributing the rate regarding the position. When X_i^j exceeds total energy of the node, the node dies.

Figure 9 shows the lifetime of the nodes in each group. As expected, group 1 has the lowest lifetime. We can also see from the figure that most of the nodes die at the same time in group 1 since a death in this group forces the remaining nodes to forward the packets of lower level nodes and die quickly. This is also true for other groups but since the loads of the first group is higher, this result is more noticeable in group 1. Group 4 nodes also die at the same time since the load can be considered same for all nodes. On the other

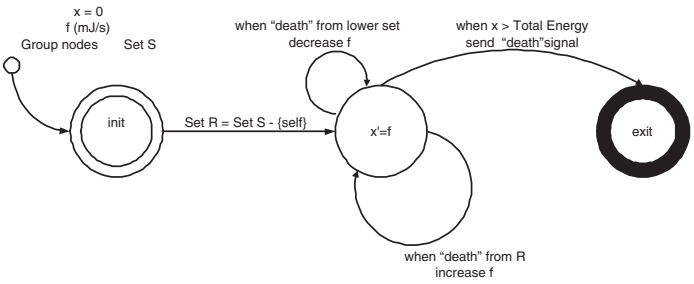

Figure 8: Hybrid Automata Model of A Single Sensor Node

hand, group 2 and group 3 nodes die gradually. That is reasonable since the load of that nodes varies and lifetime is not homogeneous.

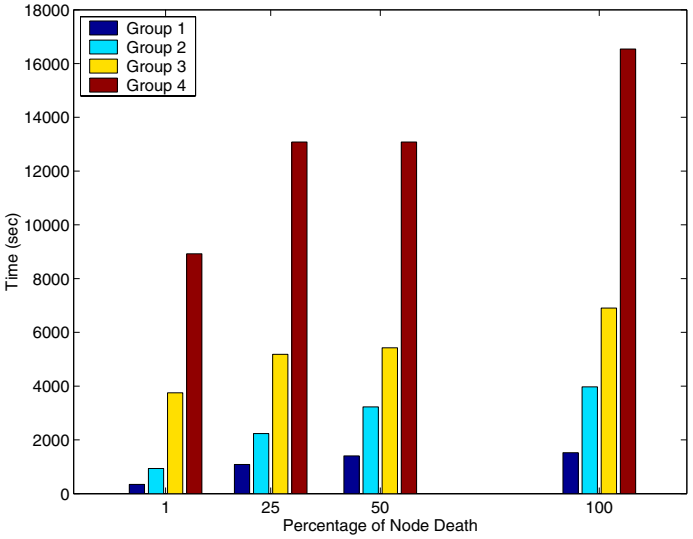

Figure 9: Performance Results for the Network in Figure 7

5. RELATED RESEARCH

Automatic verification of embedded systems is essential for safety critical applications. The safety check procedure for railroad gate controller and for a timing-based mutual exclusion protocol is performed in [9] with HyTech. In [10], the verification of the controller is performed for the safe operation of the boiler after obtaining abstracted linear models for the nonlinear behavior of the boiler again with HyTech.

Simple power analysis of a networked sensor node is performed in [5]. Power cost of instructions such as writing to memory, external functions such as sensing and overhead of basic operations in terms of clock cycles such as posting a task is measured. By using the results of this paper, we have performed automatic power analysis of the nodes as a function of the distance from the central data collector.

Power analysis of a sensor network is performed in [7] and [6]. In these schemes, the sensor network is configured in a hierarchical manner where the sensor nodes are clustered and one of the nodes is selected as the head of the group

and sends the collected data directly to the base station. In these schemes, scheduling problem arises because of the one hop forwarding and need complex algorithms which by it self, consumes lots of power. Extreme power dissipation of the head of clusters is solved by random or chain-based scheduling of the heads. Although this hierarchical scheme is a solution for small size networks, when large size networks are considered, head of clusters still needs multi-hop communication to the base station over tree based configurations. Therefore, our analysis is for tree-based multi-hop networks.

6. FUTURE WORK

Applying formal verification techniques (and ideally synthesis techniques) to sensor network systems and applications is an extremely important direction. The constrained setting and the structured environment in TinyOS make this a very attractive direction. However, the current automata model is constructed by abstracting the performance of the system with extensive human intervention. In the future, we would like to automate the abstraction procedure from the TinyOS code along with some high level performance specifications to the hybrid automata model.

7. CONCLUSION

In this paper, we analyzed TinyOS, an event driven operating system for sensor networks. We modelled TinyOS with the Hybrid Automata. We made a verification check and analyzed the power dissipation of a node which formed a tree-based multi-hop sensor network and had different rate of power dissipation with respect to its position. Following the analysis of a single node with HyTech, we modelled the sensor network as Hybrid Automata to analyze the power dissipation over the sensor network. Results show that sensor nodes near and far away the base station has the highest and lowest lifetimes respectively and a large fraction of nodes die at the same time. On the other hand, nodes in the middle have a variation in their lifetime. For a sensor network to be in operation, connection to the base station always should be alive. This requires the usage of the nodes with high energy or dense distribution near the base station.

8. REFERENCES

[1] K. S. J. Pister, J. M. Kahn and B. E. Boser, *Smart Dust: Wireless Networks of Millimeter-Scale Sensor Nodes*, Highlight Article in 1999 Electronics Research Laboratory Research Summary.

[2] J. M. Kahn, R. H. Katz and K. S. J. Pister, *Mobile Networking for Smart Dust*, ACM/IEEE Intl. Conf. on Mobile Computing and Networking(MobiCom 99), Seattle, WA, August 17-19, 1999.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister, *System Architecture directions for network sensors*, ASPLOS 2000.

[4] T. A. Henzinger, P. H. Ho and H. Wong-Toi, *Hytech: a model checker for hybrid systems*, Software Tools for Technology Transfer 1: 110-122, 1997.

[5] E. J. Riedy and R. Szewczyk, *Power and Control in Networked Sensors*, today.cs.berkeley.edu/tos.

[6] S. Lindsey, C. S. Raghavendra. *PEGASIS: Power-Efficient Gathering in Sensor Information Systems*, IEEEAC, 2001.

[7] W. Heinzelman, A. Chandrakasan, and H.Balakrishan. *Energy-Efficient Communication Protocol for Wireless Microsensor Networks.*, Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, Maui, HI, USA, 4-7 Jan. 2000.

[8] SHIFT *The Hybrid System Simulation Programming Language* http://www.path.berkeley.edu/shift

[9] R. Alur, T. A. Henzinger and P. H. Ho, *Automatic Symbolic Verification of Embedded Systems*, IEEE Transactionson Software Engineering 22: 181-201, 1996.

[10] T. A. Henzinger and H. Wong-Toi, *Using Hytech to Synthesize Control Parameters for a Steam Boiler*, Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, Lecture Notes in Computer Science 1165, Springer-Verlag, 1996, pp. 265-282