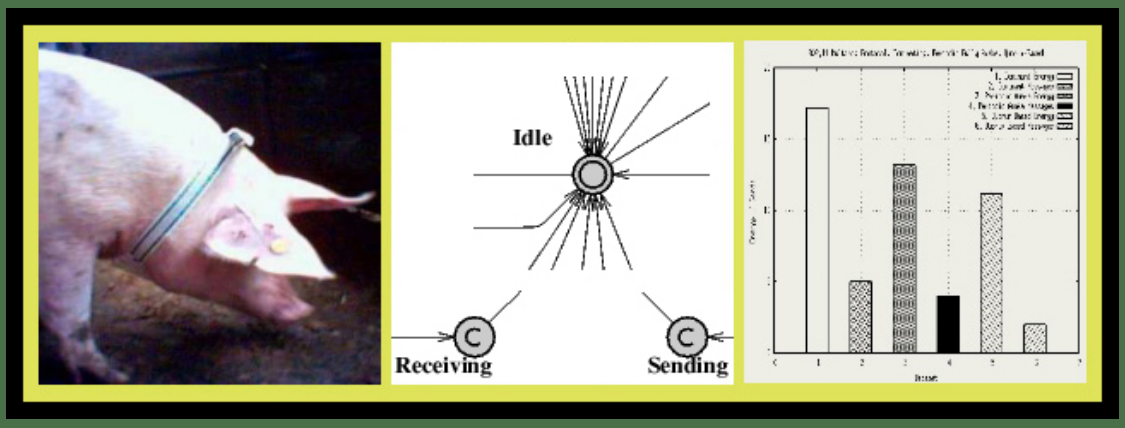
Verification of Sensor Network Models using UppAal

Troels Frederiksen Smit, s991067



LYNGBY FEBRUARY 2004
M. SC. THESIS
NR. 14

IMM

Printed by IMM, DTU

Abstract

In this report research is done on using the *UppAal* framework with relation to analyzing energy consumption in sensor networks. A model of a sensor network is created, tested and verified. Then the possibilities of formal reachability analysis examined. This results in a scenario based worst-case analysis of both total energy consumption and energy consumption patterns. A framework composed of tools for sensor network model creation and automated analysis is also developed.

The thesis is a part of the *Hogthrob* project, which goal is to develop sensor network technology adapted to the requirements of sow monitoring.

Preface

An interesting conversation between *DTU* professor Jan Madsen and the author lead to the idea of combining formal verification with models of sensor networks. The *UppAal* framework is known as a stable platform for development and verification of timed automata and was chosen as the model basis. The aim of the thesis was not to be comprehensive but to discover which opportunities the new combination of the research areas would yield.

Acknowledgments

I would like to express my sincere thanks to my thesis advisor professor *Jan Madsen* for his brilliant and always enthusiastic guidance, everyone active with the *UppAal* mailing list including *Gerd Behrmann, Dave Ashley, Andrew E. Santosa and Fernando P. Schapachnik*, and my father *Steffen Smit* for proof reading the thesis. Errors which remain have been inferred after his help and remain entirely my fault. Finally the academic environment at the middle fourth floor at the Egmont dormitory has provided continuous encouragement and support for which I am indebted.

Troels Frederiksen Smit, 28 February 2005

Contents

List of Figures

CHAPTER 1

Introduction

Sensor Networks are composed of small mobile nodes interconnected into a wireless network. The nodes consist of sensors, a radio and a microcontroller managing the activity of the node. The micro controller may be equipped with an operating system such as TinyOS, an event-based operating system, developed for sensor nodes at the University of California, Berkeley. Sensor networks are powered by small batteries. Therefore, it is critical to optimize the nodes and the programs to minimize energy consumption. The code size and execution time are also limited by the memory size and the real-time constraints. The sensor node itself may be designed and implemented as a heterogeneous multiprocessor system, i.e. a complicated System-on-Chip.

The *Hogthrob* project, which this thesis is a part of, deals with networked on-a-chip nodes for sow monitoring. Node lifetime is aimed at six month or higher. The department of Computer Science at the University of Copenhagen (DIKU) has already presented work in the area of modelling the energy consumption of the *Hogthrob* nodes.

In this project we wish to examine the possibilities of using formal analysis to verify properties of sensor networks. In particular, we are interested in reasoning about the energy consumption of the system. This will extend the work at *DIKU* to support analysis of the system corner cases. This will give a reliable estimate of sensor node lifetime.

Formal analysis requires the use of models, trusted to behave like a real system. It is therefore critical to find the correct abstraction layer for the models and to verify the models. When reliable models exist, the formal analysis of the systems can be performed; this will give the system designers a new way of analyzing their systems. The *UppAal* framework will be used for model implementation and analysis.

The *UppAal* trace utility yield not only an estimate of node lifetime but also the possibility to analyze the energy profile. If a sensor node for example is powered by solar energy, the cells will deliver a certain amount of Joules in an entire day. But if every Joule is necessary in a short time span, this is potentially fatal for the node.

This report is structured as follows. In Chapter 1 an introduction into the area of sensor networks is given, the node structure and protocols are presented and related work is commented. Chapter 2 deals with the first sensor network model named *A.N.P.* The model is analyzed and important conclusions drawn which leads to the second

model named*E.N.D* presented in Chapter 3. In Chapter 4 a framework developed for test and analysis of the second model is developed. Chapter 5 presents the tests and results from using the framework described in Chapter four. Finally Chapter 6 is a discussion of the achieved results with a larger perspective in mind, and the Chapter ends with a conclusion.

1.1 Sensor Networks

Sensor Networks is a field of research moving away from power outlets and powerful antennas. Instead battery powered technology and large amount of nodes are the topics of interest. Several limitations and possibilities immediately arise as a consequence of this design choice. There are three fields of research trying to solve the challenges posed:

- Node Construction
- Node Application behavior
- Network Topology and Protocol

1.1.1 Node Construction

Regarding Node construction it is clear that much effort has gone into constructing low and *ultra*-low power consuming components [1]. In resent time research has been done in multi-processor systems [2]. Also the choice between RF and optical transmission technology is important as discussed in[3]. What is common between sensor network nodes is that they consist of at least the following three possibly integrated parts, a radio, a chip running an operating system and an instrument.

Radio

A typical radio for sensor networks will be able to go into sleep mode where it uses extremely little energy, for instance the pico radio designed by the PicoRadio group at the Berkeley Wireless Research Center [4].

Chip and Operating System

The choice between using an *ASIC* and a regular micro controller is easy from a designers point of view. Unfortunately, *ASICs* are extremely expensive to produce and thus are almost never used in a prototype or on a test platform. As a compromise a system will often consist of a micro controller in combination with the powerful *FPGA* technology. In the *Hogthrob* project a *ATMega128l* 8-bit, RISC micro controller is interfaced with a Spartan3 XC2S400 *FPGA* from Xilinx, which together constitutes the core of the *V0* platform. The opportunities held by such a system is similar to an *ASIC*, though it will always consume more energy. Programs for the *ATMega128l* can be compiled using the GNU GCC tool chain. This allows for easy porting the open source operating system *TinyOS* to this platform. TinyOS is an event-driven and slim architecture which allows designers to aim at obeying time-critical deadlines. Unfortunately it cannot guarantee deadlines to be met like a Real-Time Operating System, but the small and fast nature makes it a frequent choice for sensor network nodes.

Instrument

To serve a purpose nodes will (almost always) need an instrument to measure their environment. The instrument may measure temperature, movement or other physical data. The instrument will often be used in a certain cycle, for instance every five minutes. Whether data processing and analysis is performed on the node or raw data is transferred, is a decision taken after evaluating the integrated computational power of the node and the power required for the data processing compared to transmitting the data.

1.1.2 Node Application Behavior

The Application behavior has been analyzed to have tremendous impact on energy conservation. Application designers will always remember to set the radio in sleep mode and not in idle mode after having read the article "Energy Aware Wireless Sensor Networks" by *Vijay et al.*[5].

Researchers respect the consequence of sending a RF signal over long distances. This has lead to protocols based on either one of the following:

- Single Hop
- Multi Hop

1.1.3 Single Hop

The case where the system transmits directly from the node to the base-station is called single hop. This system has been proposed in several protocols such as[6] and is implemented several places such as[3] (using optical technology for transmission) and [7]. The clear advantage of single hop is simplicity and the possibility of direct node communication. The basic protocol for sensor networks is the single hop based protocol. When using this protocol all nodes send their data directly to the base-station. An example of this can be seen in figure 1.1.

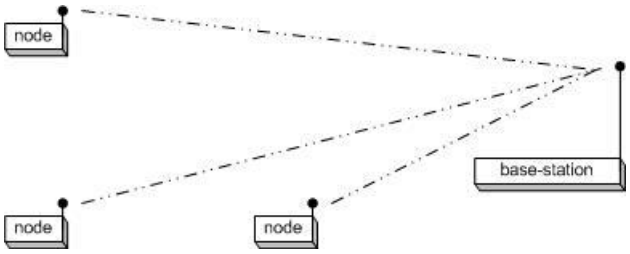


Figure 1.1 *Single Hop Protocol*

Contention for the medium used for transmission (e.g. a radio channel), is the major protocol problem that shall be solved.

1.1.4 Multi Hop

Multi Hop is used when it is either impossible or requires a prohibitive amount of power for the nodes to transmit directly to the base-station. The nodes therefore inter-transmit the messages in an attempt to approach the base-station with all information without using much power. One of the best examples of this is the *ZebraNet* project [8], where nodes are scanning for nearby nodes. Once a node is in sending distance of another node, all available data is transmitted between the nodes (zebras). Zebra researchers then need only find a couple of zebras to get information from all zebras. A simple multi hop network can be seen in figure 1.2.

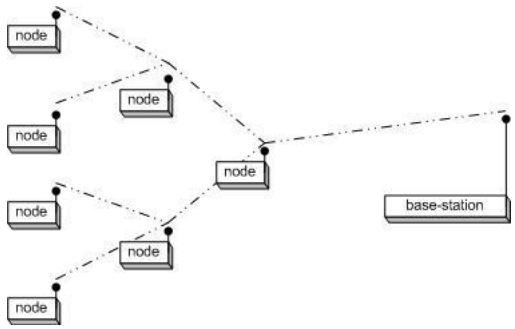


Figure 1.2 *Basic Multi-Hop Protocol*

The problems faced with multi hop algorithms expands from medium contention to also include message routing.

1.1.5 Problems faced with Sensor Network Protocols

Sensor network protocols must face and resolve several important problems, these problems include:

- Contention for transmission medium
- Multi-Hop Routing

In the following subsections these problems are explored.

1.1.6 Contention for transmission medium

The Hidden Terminal Problem

An example of the hidden terminal problem is illustrated in figure 1.3. The scenario is that node A and node C is trying to send a message to node B at the same time. Node A and C will both determine the network to be free and start transmitting. The problem is then that the node B will never receive any signal, since node A and node C unknowingly will create a mutual block.

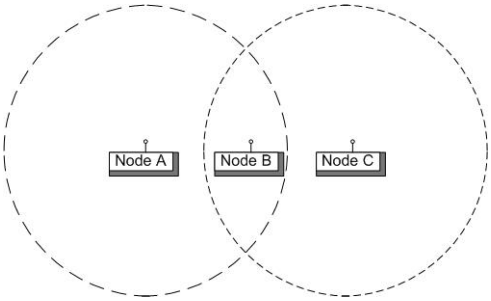


Figure 1.3 *The hidden Terminal Problem*

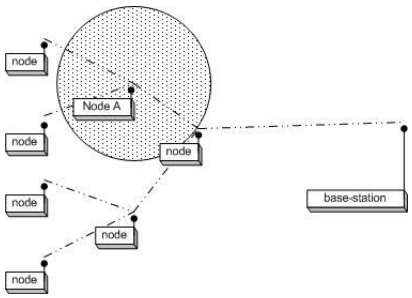


Figure 1.4 *Overhearing, lowest transmission strength*

Overhearing

When a node uses Radio Frequency signals in *ad hoc* networks to communicate it does not know where the receiving node is placed. The result is that the signal must be equally strong in all directions, thus conforming to theory behind an isotropic antenna. As a consequence, nodes might by accident overhear messages not destined for them. The efforts taken to overcome this problem include varying the transmission strength and introducing transmission schemes, inferring the need for synchronization. These problems or challenges may exist for both single and multi-hop protocols.

Overhearing, lowest transmission strength

Using a very low transmission strength the network might be very fortunate and only the receiving node be affected by the transmitted signal. An example of this can be seen in figure1.4 where the signal exactly reached the antenna of the rightmost node.

Overhearing, medium transmission strength

Unfortunately it may occur that nodes are overhearing the transmission. An example of this can be seen in figure1.5. When a node is using its radio to receive a signal not destined for it, it can be consider throwing important joules out of the window.

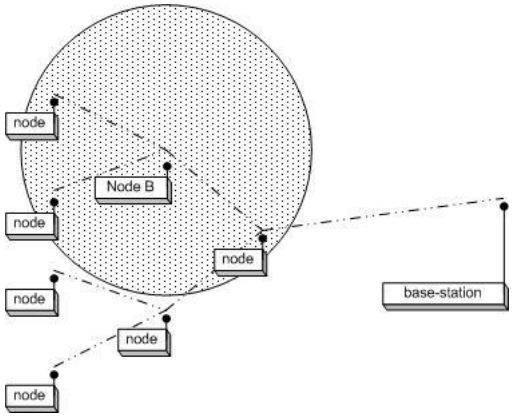


Figure 1.5 *Overhearing, medium transmission strength*

Overhearing, strong transmission strength

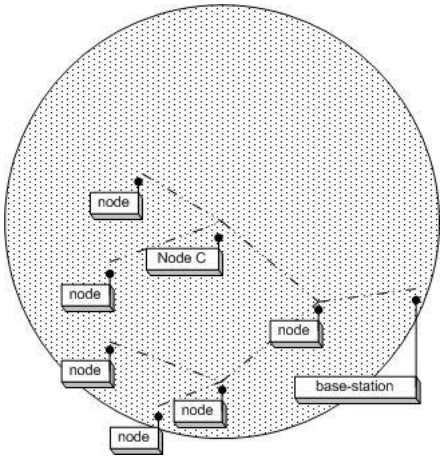


Figure 1.6 *Overhearing, strong transmission strength*

Using a forceful transmission strength have devastating consequences. This is illustrated in figure 1.6. The transmission from node C is overheard by eight nodes, of which many could otherwise have communicated themselves.

Synchronization

Synchronizing the nodes will be a great benefit for protocol simplification. Unfortunately, this is rarely possible in *ad hoc* networks, and therefore many protocol designers does not include this option in their protocol.

1.1.7 Multi Hop Routing

Using a multi-hop protocol does not solve the problems faced by the single-hop protocols. It only solves the potential energy waste from transmitting signals over long distances. On top of the single-hop protocol challenges new problems arise:

Defining a Route for each message

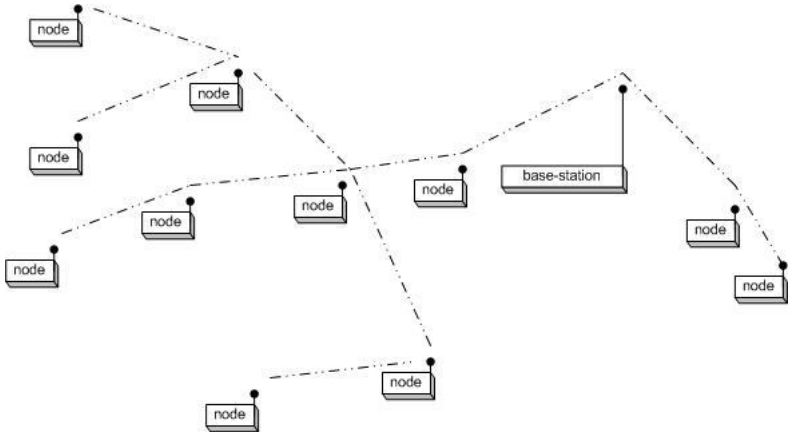


Figure 1.7 *Routing*

Defining a route from node to base-station is a major issue. It is extremely complicated to find the optimal route; hence *ad hoc* solutions have been proposed.[9][10]. In figure 1.7 an example of a routed network is shown.

Neighbor Discovery

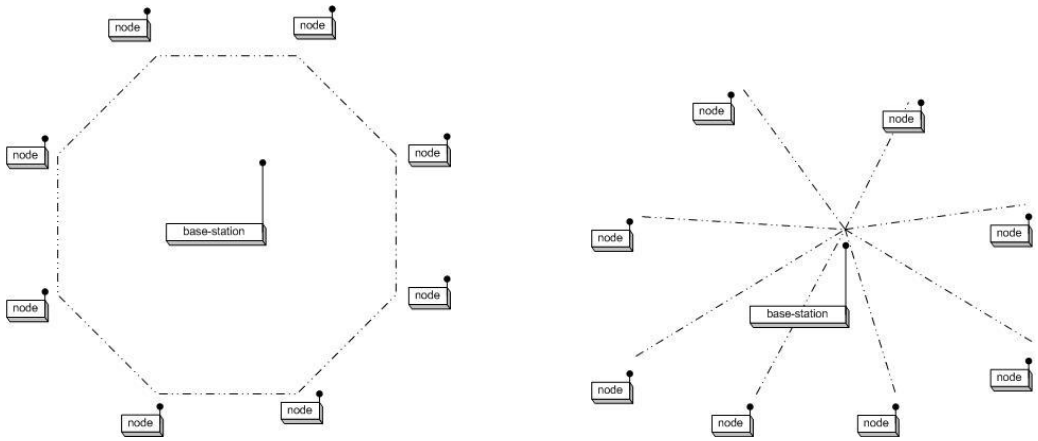
Creating a routed network first requires that nodes know which neighbors are in listening- and transmitting within distance.

Network Partitioning

When all nodes know which nodes they may communicate with, the network can be partitioned.

Equality Path Problem

One of the sub-problems of creating a routing for node messages are the equality problem. All nodes are imagined placed with the same distance to the base-station. Several solutions exist for the manner in which the messages should be passed to the base-station. As is shown in figure 1.8(a) and 1.8(b). Which solution is best depends on the message length, node composition (buffer size), timing etc.



(a) Node Routing setup in Circle

(b) This is in square

Figure 1.8 *Determining the optimal route*

1.1.8 Protocol types

Wireless sensor networks MAC protocols are divided into Contention based and Time Division Multiple Access protocols.

Contention Based Protocols

Contention based protocols include: *802.11, PAMAS and MACAW* [11][12]. When using a contention based protocol, the problem of scheduling is not an issue. Unfortunately, presented problems such as *The Hidden Terminal* becomes important, especially in dense networks.

TDMA Based Protocols

Protocols based on TDMA includes the *MIT Leach*[9]. One of the advantages of *TDMA* is energy conservation because of small duty cycles. These protocols require that a scheme for scheduling is inferred. The scheduling in turn scales well when the number of nodes increase, especially when using cleverly designed clusters.

1.1.9 Event Streams

When the problems of transmitting data correctly have been resolved by a protocol, the result is a network with specific characteristics. Seen by the nodes these characteristics can be modelled as incoming, outgoing and passing traffic. The traffic seen by the node can thus be modelled as shown in figure1.9.

The problem of simulating the behavior of a node is then a matter of generating event-streams.

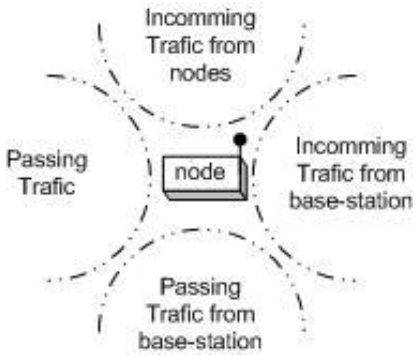


Figure 1.9 *Node in with incoming types of Traffic*

Events streams has successfully been used to model sophisticated systems in[13]. The novel approach is thus the manner of generating the event streams and research done towards this end. Especially establishing the correct abstraction layer for the models and verifying this is a long step forward towards holistic and correct formal modelling. The event streams should correctly reflect changes in network behavior as consequence of design decision. This will yield a model capable of finding the worst-case behavior of a network topology operating under a specific protocol on a specific platform. The necessary information to correctly model a Sensor Network thus include information on the following subjects:

- Network Topology and Node position
- Communication Protocol
- Node Platform

Exploration on the use of event-streams is done in chapter 3 on the *E.N.D* model.

1.2 Estimating Energy Dissipation

There are several ways to model the energy consumption in a node or sensor network. To appreciate the pros and cons of formal analysis it is necessary to know the alternatives, and thus a quick summary is now given.

1.2.1 Analyzing energy dissipation

Article [14] on *Lifetime Analysis of a Sensor Network with Hybrid Automata Modelling* is closely related to what is proposed in this thesis. The authors used the *HyTech* automatic tool for the analysis of embedded systems. HyTech is capable of computing the condition under which a linear hybrid system satisfies a temporal requirement, very much like *UppAal*. The important thing to notice is that they only use the model checker for system verification, and then turn to the program language *SHIFT*, capable of describing dynamic networks of hybrid automata, for all purposes of analysis and thus looses the power of formal analysis.

Article [15] introduces *Simulating the Power Consumption of Large-Scale Sensor Network Applications* presenting *PowerTOSSIM* which is capable of simulating the energy consumption of each node in a sensor network based on the TinyOS operating system. It is based on assigning energy consumption to the different node actions.

In [16] on *Power Estimation using the Hogthrob Prototype Platform*, a VLSI design of a node is analyzed. Martin Leopold does so in two ways. First using the *Synpsis Power Compiler* and secondly using an abstract Power model for each component of the *SoC* node is proposed and used *p.56*. In the master thesis M. Leopold recognize the necessity of traces and put an effort into analyzing how traces of system behavior can be obtained.

The *SENS, Sensor, Environment and Network Simulator* presented in[17] is a customizable sensor network simulator for wireless sensor network application. It is possible to exchange the system components, and the simulator facilitating a diagnostic facility for power utilization analysis. The emphasis is on the environmental impact on sensor network simulations and provides simulation in a fashion similar to TOSSIM. An API allows easy integration with for example TinyOS programs, which can be compiled and executed on a work-station.

For sensor network node behavior, article [18] presents accurate and scalable simulation of entire tinyOS applications.

Finally in article [19] the authors have obtained their results though an application they have developed themselves. This is a very precise way of modelling, but ineffective since they are *"reinventing the wheel"* and may easily fail or forget to implement important design issues.

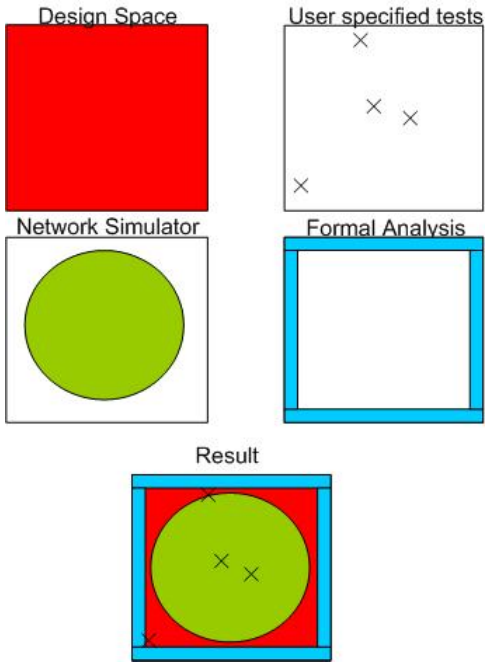


Figure 1.10 *Analysis and Verification of sensor networks*

1.2.2 The formal approach to analyzing energy dissipation

One of the strength in formal analysis is in the exhaustive search for reachable states. If the designer of a system uses a simple script simulating his environment, only a very small part of the design space and analysis space will be covered. Using test environments that have already proven their worth such as *PowerTOSSIM* and *ns-2* will allow for a much larger space to be covered. What is left is the corner cases, which figure 1.10 demonstrates. Using regular methods of analysis it is almost impossible to reach the corner cases. It can be extremely difficult to imagine the required system interactions to reach the state and hence impossible to test. Formal analysis allows the designer to ask questions such as: *Will this ever happen?* Whereas regular testing only allows for: *What happened?* On the other hand, formal analysis cannot determine *What will usually happen?*. The strength and place of formal analysis should now be obvious. A negative side of formal analysis is that the cases found may not reflect the natural system behavior, but finding the *Worst Case* energy consumption in a scenario has at least two important uses: In a resource limited and critical system worst case behavior must always be covered. Also in the process of optimizing a design the *Worst Case* behavior may yield which parts of a system should be redesigned. It is therefore concluded that corner case analysis through formal modelling indeed have an important place in future analysis and verification of sensor network systems.

1.3 Simulation

New protocols, system topologies and node behavior must be modelled and verified on a system scale before actual implementation can begin. Several testing environments exist for this purpose including *ns-2*. For sensor networks it is paramount that the limited energy consumption is not depleted and therefore it is interesting to incorporate the node energy consumption into the system test. Also specific to the sensor network area is the limited focus on throughput. In this section the advantages of the *ns-2* simulator is described and compared with the newcomer: the *UppAal* framework.

1.3.1 Network Simulator 2 (ns-2)

An excellent environment for testing network simulations is the Network Simulator - ns2 [20]. Ns is a discrete event simulator which provides support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. Several network building blocks have been developed ensuring that building the network scenario is fast. These building blocks include:

- One-way TCP (Tahoe, Reno, Vegas, SACK) [21]
- Scheduling algorithms: SFQ (Stochastic Fair Queueing), FQ (Fair Queueing) and DRR (Deficit Round Robin Scheduling).
- Lossy links
- Support for mobile hosts

The *ns-2* simulator is thus an already proven and very efficient network simulator widely used.

One problem with simulators, such as ns-2, is manual scheduling of events. The designer will thus have to define not only the testing scenario but also the exact time of each event. Example: *ns at 8.1 "$wireless start"* and *ns at 8.7 "$wireless stop"*. A major part of the testing work is therefore to create realistic traces of the traffic which the network will actually experience. This often dull and difficult process makes it a favorite fast-forward point in many design processes. Therefore the corner cases of the system is rarely found, and even more rare, proven to be found.
To explain the problem: A node designed to discover activity could be analyzed to work for two years in a wide range of scenarios. Unfortunately the place, where the node is situated in the network, has a lot of very unfortunate scheduled passing traffic and the node dies after two months. In the ns-2 simulation this corner case of passing traffic, with a specific unfortunate schedule, might have been almost impossible to discover.

1.3.2 UppAal

A first class tool for verifying timed system models is the *UppAal* framework [22]. The framework builds upon the theory of timed automata but has been extended to support features required by the real-time system and protocol verification community. The

basic syntax for the TCTL formulas which are sought verified are introduced in figure 1.11 from the *UppAal tutorial* [22] which also includes more information.

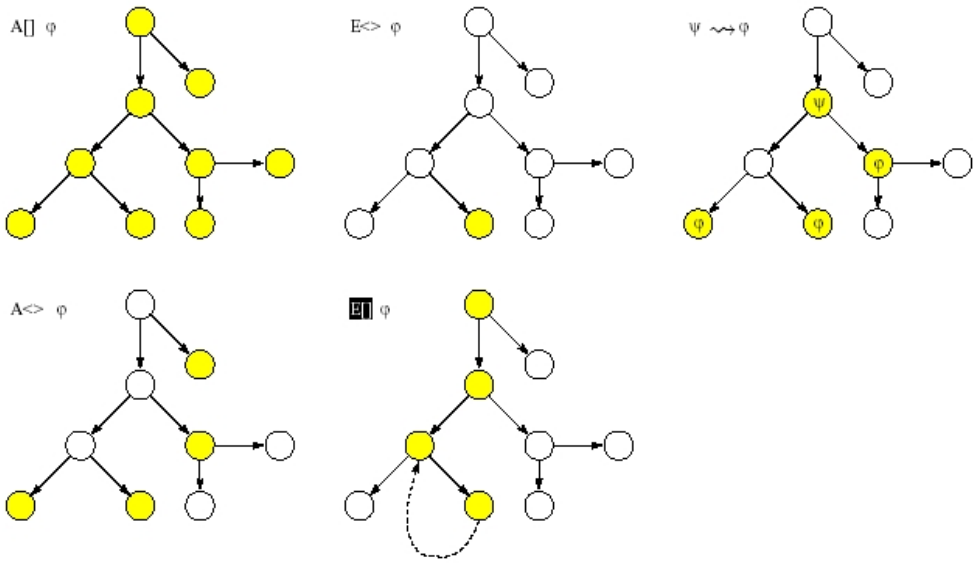


Figure 1.11 *The TCTL formula syntax*

Using the TCTL syntax the framework will answer the formulas with the precision of formal analysis. Hence protocol designers implement their model and ask questions regarding safety properties such as *Can the system ever deadlock?*

$$A[]\, not\, deadlock \tag{1.1}$$

and given the answer *satisfied* they do not have to ever fear that this will ever happen. Similarly, liveness properties such as *Will the system always try to send a message* can be asked and verified in the same manner:

$$E[]\, message.sent == 1 \tag{1.2}$$

People dealing with timing problems in real-time systems can also use the *UppAal* framework. They will seek different answers though, and ask questions such as:

$$E <> MessageReady \&\& SystemBusy \tag{1.3}$$

or

$$E <> Train1.ChosenRail == Train2.ChosenRail \tag{1.4}$$

performing an exhaustive search for the destructive state, which the system should never reach. The *UppAal* framework is therefore an extremely versatile tool that has already proven it's worth for large corporate organizations, but still has much potential left to explore.

1.3.3 Wishful Thinking

The power of the *ns-2* network simulator is weakened by the hard work required to generate reliable event scheduling, and exhaustive search for the worst case scenario is often impossible. Wouldn't it be great if is was possible to relieve this exact problem? Wouldn't it be great if you could ask: *"If I put the system in this scenario how much energy will maximum be dissipated?"*. In this thesis an attempt is made to implement a sensor network model using the *UppAal* framework. The pros and cons of this attempt will be mapped, and hopefully we will end up finding the X in the following questions:

$$\begin{aligned} E <> CLOCK < 1000 \,\&\&\, Energy > \quad X \quad (true) \\ E <> CLOCK < 1000 \,\&\&\, Energy > X+1 \; (false) \end{aligned} \tag{1.5}$$

Which translates to: *In the current scenario, does there ever exist a state where the clock has not passed 1000 time units and the energy consumption can be X but not X+1.*

1.4 Related Work

This is a discussion of work related to this report. It positions this report among the hectic ongoing research regarding sensor networks and formal verification of models.

Regarding simulation and modelling of sensor networks, effort has been put into modelling energy consumption, and low power node construction is therefore well understood.

1.4.1 Sensor Network Applications

Prototypes of large sensor networks have already been created. Amongst these are the Zebra project[8] the Great Duck Island project[23], the Hogthrob project [24] and the non-RF based project described in[3].

1.4.2 Simulation / Modelling

Other research has been dealing with the time required to simulate a large number of nodes[25]. The limitations of these very detailed and thorough simulations is the extraction of the needed information. To determine whether a system deadlock, exhibit liveness or has extreme energy peaks in the energy consumption pattern can be extremely difficult in these environments. This is why formal verification is such a hot subject.

In article [26] on "Timed Autometa: Semantics, Algorithms and Tools" and article [27] about “Timed vs Time Triggered Automata” the semantic foundation and theoretical background for the *UppAal* framework program is described.

An interesting SoC/NoC framework which yields good potential for formal analysis is introduced in [2]. The real-time characteristics of a system is discussed in[28].

A project doing analysis on the correctness of a TinyOS model using *HyTech* instead of the *UppAal* framework has been done in[14]. They use *SHIFT*[29] to estimate the average lifetime of a sensor node, but *HyTech* is only used for protocol validation.

In[13] the idea of using a formal approach to Multiprocessor System on Chips (*MpSoC*) performance verification is presented. The paper has several important points: *”System-Level performance verification is one of the top three codesign issues”* and *“Simulation-based performance verification, however, has conceptual disadvantages that become disabling as complexity increases”* and *“finding simulation patterns - or use cases - that lead to worst-case situations is challenging”*, *“formal analysis guarantees full performance corner-case coverage and bounds for critical performance parameters”*. They perform *“Process execution time analysis and Scheduling analysis”*. The idea of event streams is introduced.

1.4.3 UppAal

Regarding verification of models, much effort has been put into construction of the framework *UppAal*. The *UppAal* framework has already proven its unique qualities in

the area of Real Time Constraints Validation as presented in article[30] by Hongyan Sun. Sun validates several basic Real Time system scheduling policies such as can be found in the book "Real-Time Systems"[31].

This report is not concerned with real-time constraints. Instead it estimates power patterns. This is a new approach to the *UppAal* framework which has not been taken before.

The *UppAal* framework uses an advanced model of computation[32] and the *UppAal* crew has created an introduction to using the *UppAal* framework efficiently and correct[22]. Models for computation is not limited to the methods of the *UppAal* framework, and a introduction to this subject can be found in[33].

In[34] about "Compact Data Structures and State-Space Reduction for Model-Checking Real-Time Systems", the effort taken to reduce the state space in the *UppAal* framework is described. The *DBM* Difference Bounded Matrix data structure, which offers a canonical representation for constraint systems, is mentioned with the work extending this data structure. They note that required space is a increasing as O^2 with the number of clocks and the algorithm speed is running O^3 with the number of clocks. Thus, minimizing the number of clocks is critical. [35]

1.4.4 Protocols

The most commonly used protocol for single channel communication in todays industry is the ALOHA protocol presented in[36]. The aloha protocol was expanded to the Time Slotted ALOHA protocol as described in[37].

The S-MAC protocol presented in[38] is the a mature choice for sensor network communication supporting multi-hop. It identifies the major source of energy waste in a sensor network communication as: *collision, overhearing, control packet overhead and idle listening*. The authors use the power relation *idle:receive:send* ratio to compare protocols. The authors introduce *adaptive listening*, taking the role of the *idle* state, enabling *Overhearing Avoidance*. The listen-interval is the duty-cycle percentage of the frame (with 10% duty cycle i.e. 115 ms, then the frame is 1.15 s). The authors conclude that periodic sleeping provides excellent energy performance at light traffic load, but adaptive listening is able to adjust to traffic and provide energy performance as good as no-sleep at heavy load.

The S-MAC builds on research on wireless LANs which in 1994 resulted in the MACAW protocol presented in[12]. The most important results are obtained by use of *CSMA* technology, where the surrounding signal strength in the vicinity of the transmitter is measured. This allows the protocol to defer transmission if the sending medium is concluded occupied. The hidden terminal problem then arises, but in general less contention for the transmission medium is gained.

In[6] *a New Protocol for Low Power Sensor Networks* is introduced. It is a single hop MAC protocol, which use the idea of network processing and evaluation in combination with only having nodes initiating communication to obtain low overhead for communication. The backside is the need for synchronization and the potential long transmission distances for the nodes. The article authors are aware of and working on

a solution to these issues.

The article on *PAMAS*[11] correctly identifies the problems in creating an efficient protocol for wireless sensor networks. They create the *PAMAS* protocol by merging the ideas in [12] and the idea of using a separate signalling channel as introduced in [39]. They obtain several results which I will try to verify using the *E.N.D Model*

Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks introduced in[19] gives a strong insight into the area. The article list the solution strategies as follows:

- Transmission Power Control
- Power-Aware Routing
- Low-Power Mode

Furthermore the challenges are listed as:

- Clock Synchronization
- Neighbor Discovery
- Network Partitioning

They introduce three protocols for WLAN multi-hop. For verification purposes they measure three metrics:

- Power Consumption
- Power Efficiency
- Neighbor Discovery Time

They define the event distribution to describe the event-stream which is a very effective way of simulating a complex environment.

The article[9] on Energy-Efficient Communication Protocol named *LEACH* for Wireless Microsensor Networks, is a part of the MIT μ-amps research project[1]. They introduce a protocol for the typical Sensor Network scenario:

- The base station is fixed and located far from the sensors
- All nodes in the network are homogeneous and energy constrained

They introduce the following concepts in the *LEACH* protocol:

- Localized coordination and control for cluster set-up and operation
- Randomized rotation of the cluster "base stations" or "cluster-heads" and of the corresponding clusters.
- Local compression to reduce global communication

The nodes take turn in being cluster heads and doing tough energy transmissions. The authors compare their protocol with the single hop and a static clustering protocol.

In[40] *"GPSR: greedy perimeter stateless routing for wireless networks a protocol using Distance Vectors, Link States and Path Vector routing algorithm"* is introduced. Information on geography is also used in[10] *"Geography-informed energy conservation for Ad Hoc routing"* and [41] *"Span: An energy-efficient coordination algorithm for topology maintenance in Ad Hoc wireless networks"*.

1.4.5 Application Challenges

Frequently "Tiny OS" has been used for simulation[15].

Amongst the best articles discussing real measure of energy consumption is[5] which in detail describe *"where the power goes"*, so future research have been guided in the right direction.

The paper[42] *"Building Efficient Wireless Sensor Networks with Low-Level Naming"* deals with in network processing and data aggregation for network traffic reduction. The idea is to give nodes names, reflecting properties, such as their geographic position or abilities if using heterogeneous nodes. The authors verify their model on a testbed with 14 PC/104 sensor nodes.

In article[43] on *"Multi-dimensional range queries in sensor networks"* they use the GPSR[40] protocol to enable multi-dimensional range queries such as "List all events whose temperature lies between 50° and 60°". A distributed data structure is introduced called "DIM". This data structure efficiently resolve multi-dimensional range queries. Another work in this area is the *"Networking support for query processing in sensor networks"* presented in [44], which deals with the benefits from two-way communication.

Other work in the area of sensor networks deals with signal safety and integrity [45] and the feasibility of new ideas such as energy harvesting [46].

1.5 Summary

The lifetime of sensor network nodes is recognized as vital information for almost all sensor network application purposes. Lifetime information has been modelled and approximated through simulation using simulating environments such as *PowerTOSSIM, SENS and ns2*. The best and worst corner cases have never been examined because they are extremely hard to find and can never be proven correct using the mentioned simulation environments. If a designer has a battery with a certain capacity it is important to know if it can be promised to be enough for e.g. 6 month or two years. Formal modelling can answer this question.
There are two approaches to formal modelling of a sensor network. The first approach is to model the entire network as done with *ns2*. This way of modelling means, that an *ns2* model could be directly translated and thus create a unified framework for simulation and verification purposes. A potential problem is the extremely large simulation space, which cannot be verified on a regular workstation computer. This way of modelling is examined in Chapter 2.
The other approach places a target node as the center of the test. The surroundings are then modelled through event streams upon which the node will then react. This creates a relatively slim simulation space and allows for the same analysis of node behavior as the first approach. This way of modelling is examined in Chapter 3.

CHAPTER 2

The A.N.P Model

2.1 Introduction

In this chapter the properties of modelling an entire sensor network is examined. This is similar to what is done with simulators such as *ns2*. The system uses a protocol proposed by Mario Neugebauer and Klaud Kabitzsch in the spring of 2004 [6]. It is called: *"A new protocol for Low Power Sensor Networks"* and will be referred to as the *A.N.P* protocol.

The chapter is structured as follows. First the *A.N.P* protocol which will be modelled is introduced. Then the report continues with an introduction to the *UppAal* model in terms of partitioning and implementation. Then a basic, but formal, analysis of the sensor network model is performed using the *UppAal* framework. This will lead to a formalized comparison of the *Time* and *Delta* protocol, as was done in the original article on the protocol. The report then moves to a discussion on what further possibilities modelling in the *UppAal* framework has and shortcomings are also discussed. Finally, a conclusion is given.

2.2 A New Protocol for Low Power Sensor Networks

An interesting article by Mario Neugebauer and Klaud Kabitzsch was published in the spring of 2004 [6]. It deals with the opportunity of creating a better protocol for Low Power Sensor Network Communication by interconnecting and analyzing the layers in the OSI model for this specific application. Some important observations were done by the writers:

- If a sensor is monitoring an environment which sometimes is changing fast and at other times are changing very slow, it is probably a good idea to only have the sensors send information when the content of the message can be considered "new" or "important" as opposed to sending at a regular interval. Sending being much more power consuming, than measuring.
- When sending a message between node and base-station becomes a rare event, more nodes may share the same channel with a low chance of mutual attenuation.
- All communication can then be initiated by the nodes

A MAC scheme, which enables this protocol is reported in the article. In short it can be described as giving each channel a sub-part of each MAC cycle to do its communication.

2.3 UppAal Model - Partitioning

The number of models running concurrently is the obvious basis for an exploding state-space. It is therefore very important to keep the number of working models to an absolute minimum, while still performing a correct modelling of a situation.

In this section the layout of the model will be described, and the reasons for the choices done towards this end will be discussed.

The protocol described in Section 2.2 identify the issues with communication between the "nodes" and the "base-station". In this report we model more than that. The *system mode* is composed of several models running concurrently. The nodes will be modelled through the *Single Node (Delta/Time)* and the *base-station* modelled through the *Basestation Model*, but also the *environment* will be considered. The environment have more to it than first expected, for example electromagnetic wave should be considered an environmental variable equal to other variables of nature. The possibility of an electromagnetic waves succeeding in sending a signal from node to base-station is determined in the *Network Model*. The environmental, which the sensor network measures, is introduced in the *Environment Model*. Finally, for analysis purposes the *Monitor Model* is inferred. Each of these models are attached in Appendix C.1 through C.6.

2.4 UppAal Model - Implementation

In this section the implementation of the models presented in subsection 2.3 is described. Once the model is well understood, formal analysis is presented in the section 2.5.

2.4.1 Global Variables

The global variables are extremely important, since they describe the topology of the sensor network. They are defined as follows for the normal system used in the analysis. The normal system contains: two nodes, two protocol channels, one network model, one environment model, and a single base-station.

Global Variables

```
1 gMaxChannels 2;
2 gIntervalBetweenMeasurements 10;
3 gMaxRetryAttempts 6;
4 gMACcycleLength 10;
5 gMaxNodes 2;
6 gTsub 2;
7 gMaxSentMessages 1;
8 const gEnvironmentChange 5;
```

Understanding the model will be much easier after an introduction to the use of each global constant:

- The $g_{MaxChannels}$ defines how many channels the base-station has at its disposal - besides the special channel used for initializing nodes.
- The $g_{IntervalBetweenMeasurements}$ defines how often the nodes will try to measure the environment. This works both for the time and the delta protocol case.
- The $g_{MaxRetryAttempts}$ determines how many times a node, in vain, will try to contact the base-station.
- The $g_{MACcycleLength}$ defines the number of time-slots in a MAC cycle.
- The $g_{MaxNodes}$ is the number of nodes which the base-station will communicate with.
- The $g_{T_{sub}}$ is the number of time-slots it takes for the node or base-station to perform a single setup and transmission of a message.
- The $g_{MaxSentMessages}$ is used to create a bounded variable counting how many messages a node has sent.
- The $g_{EnvironmentChange}$ variable defines how often the environment has changed more that *delta*. This is necessary to model the *delta* protocol correct.

Now moving to consider the global variables. These are important because they, together with the *committed* state type, form the basis for safely sending variable between models. Notice also that all variables in the entire model are bounded as hard as possi-

ble. This is done because a variable with many possible values takes much longer time to evaluate than a variable which may only assume few different values. The global variables can be seen in the following code listing:

Global Variables

```
1 int[0,gMaxChannels] gChanNumber :=1, gTryingToUseChan;
2 int[0,1] gTakenChan[gMaxChannels+1];
3 int[0,gMaxNodes] gOweResponse[gMaxChannels+1];
4 int[0,1] gEnvironmentDataAvailable[gMaxNodes+1];
5 int[0,gMaxNodes] gTotalOweResponse;
6 int[1,gMaxNodes] gSendID,gSenderID;
```

- $g_{ChanNumber}$ defines which channel the base-station is currently using.
- $g_{TryingToUseChan}$ defines which channel a node is trying to use.
- $g_{TakenChan}$ is an array which defines if a channel is taken or available.
- $g_{OweResponse}$ is a boolean array which knows which nodes successfully has sent a message to the base-station. The base-station will try to respond to each of these.
- $g_{EnvironmentDataAvailable}$ This array has a slot for each node. If the environment model has set the slot to 1, then the node will perceive the environment to have exceeded *delta*. The node then resets the value to zero.
- $g_{TotalOweResponse}$ is the total number of responses the base-station has not yet answered.
- g_{SendID} is a channel ID which is sent between the base-station and node model.
- $g_{SenderID}$ is a node ID which is sent between the node and the base-station model.

Finally the *UppAal* framework model requires that we define the channels with which the models communicates:

Channels

```
1 chan initiateNodeReq, useChannelSet, useNetwork,
2      channelAvailable, failed,BaseReceive, BaseSend,
3      gFreeNet,nodeMonitor,envMonitor, nodeInit,
4      measure,sendMessage,startup;
```

And the global clock which will be used for synchronization.

Global Clock

```
1 clock gClk;
```

With all the global data constructed, we can now consider the implementation of the individual models.

2.4.2 Single Node Delta Protocol Model

Modelling an entity such as a Sensor Node is a complex and time consuming task. To make the understanding of the idea behind the implementation easier, the Node model has been divided into five different sub-partitions. This can be seen in Appendix C.1 by the dotted lines. The sub-partitioning is as follows:

1 Initializing.

2 Measuring.
3 Sending to base-station.
4 Waiting for base-station response.
5 Fail/Abort current action.

Initialization Procedure

When the node is in its initialization state it has either never communicated with the base-station, or has given up doing so on the currently assigned channel. It will now attempt to connect to the base-station using channel zero and will expect to receive the channel on which it should send its future messages. The other four states in this sub-partitioning accomplish this. If this procedure fails, sub-partition *Fail/Abort current action* is entered.

Measuring Environment

The node enters this mode "sleeping", but wakes up to measure the environment every $g_{IntervalBetweenMeasurements}$. The *Environment Model* sets the values of the global array $g_{EnvironmentDataAvailable}[nodeID]$ to '1', if the environment has changed. The nodes checks this, and only tries to contact the base-station in this case. Upon perceiving the environment as interesting, i.e. it has changed, the node synchronizes with the base-station and tries to send its message, and thus entering the next sub-partitioning.

Sending to base-station

In this part of the model, the system repeatedly tries to contact the base-station. If it fails to do so, more than $g_{MaxRetryAttempts}$ the nodes enters the *Fail/Abort current action* part of the model.

Waiting for base-station response

The node now sleeps until it knows a message from the base-station should be sent. If this message is received the cycle has ended and the nodes re-enters the *Measuring* sub-partition. Then the node will enter the *Fail/Abort current action* sub-part if it does not succeed.

Fail/Abort current action

This part of the model is used every time something unexpected happens. If this should happen, this model part makes sure that the nodes enters the system again gracefully, as described in the protocol, Section 2.3.

2.4.3 Single Node Time Protocol Model

The Node obeying the *Time* Protocol is very similar to the *Delta Node*. The difference is in sub-partition 2 - *Measuring*. Instead of checking if the environment is interesting, the

node measures the environment value and will then transmit every time an interval of $g_{IntervalBetweenMeasurements}$ has passed since the node entered sleep mode (well initialized).

2.4.4 Base-Station Model

As we saw in subsection 2.4.2 explaining large models becomes much easier upon sub-partitioning it. Hence, this is what is done here as well. The sub-partitioning of the base-station is as follows:

1 Initialize a Node
2 Receive a Message from a Node
3 Send Responses to Nodes
4 Reset Timer

Initialize a Node

This part of the model listen for nodes communicating on channel zero. It responds to the node with the channel which the node should use for future communication as well as a node ID for its messages.

Receive a Message from a Node

When a node tries to send a message to the base-station, the base-station will react using this part of the model. The variables describing the number of messages not yet replied is updated. Once this has been done, the model returns to the Initial state.

Send Responses to Nodes

This is a very interesting and moderately complex part of the model. It simply tries to respond to every node that has previously left a message at the base-station. It keeps the sub MAC cycle structure using the $g_{SubTime}$ to switch between the channels.

Reset Timer

This part of the model simply control the global clock.
This concludes the functionality of the base-station model. Which only leaves three small and easy models to be explained.

2.4.5 Environment Model

The Environment Model is controlling how often the environment is interesting for each node using the $g_{EnvironmentDataAvailable}[]$ array. It is the heart of the *Delta* model.

2.4.6 Network Model

The Network Model determines whether a signal will reach the destination, or not. If the channel is not already in use, the signal may reach the destination, but the chance of random failure is still possible. If the channel is in use, the transmission will fail.

2.4.7 Monitor Model

The Monitor is used to verify different time parameters. It is initialized every time a node successfully has initialized, and can then for example measure the time from the node starts its measuring cycle to when a message is sent.

2.5 Uppaal Model - Analysis

In this section an analysis of the sufficient relations that must exist for the model to work is given. This will validate the network protocol giving the basis for trust in the results obtained in section 2.6. First we will define that the system should never *deadlock*, but should exhibit the *liveness* property, and elaborate on the *fairness* property of the protocol. Then we will give the sufficient relations regarding both the invariants and the already introduced system constants.

It is obvious that the system must never *deadlock* and therefore the following invariant must be satisfied:

$$A[]\ not\ deadlock\ (true) \tag{2.1}$$

Liveness means that if the environment is interesting/or probed, the node will try to send a message. *Liveness* not promise that the message will be delivered, and that the protocol is fair, nor does it have any real-time constraints on delivery time. *Liveness* merely states, that the node will try to send the message. The following two relations prove that the system exhibit*Liveness* if true. The system will try to send a message:

$$\begin{aligned}(&SNDelta1.Measuring\ \&\&\\ (&gEnvironmentDataAvailable[SNDelta1.ID] == 1)\ \&\&\\ &gClk < gMACcycleLength) -->\\ &SNDelta1.Wait2Send\ (true)\end{aligned} \tag{2.2}$$

and it may do so successfully:

$$E <>\ SNDelta1.SuccessfullySent\ (true) \tag{2.3}$$

Thus, the system therefore fulfills the liveness property.

Regarding *fairness* it should be noted, that the base-station and the environment does not result in a fair protocol, as whether the signal being received by the base-station is the only priority. For the protocol to exhibit *fairness*, then every node should have a fixed time-slot to send its message, thus its signal wouldn't be interfered by nodes which happen to lie much closer to the base-station. This would make the protocol unconditionally fair - but kill the idea behind the *delta* "power saving" protocol.
Moving to consider what should be true about the system. Since the node might never initialize because of poor transmission or other connection problems, the following equation stating that, the node will always reach the state *Sleeping*, will be false:

$$A <>\ SNDelta1.Sleeping\ (false) \tag{2.4}$$

equally the node may never reach the *SuccesfullySent* state:

$$A <> SNDelta1.Sleeping\ (false) \tag{2.5}$$

It should be noted though, that these equations won't be true for any wireless network, since QoS cannot be promised.
The next part of the analysis will work with interesting states, that indeed is reachable. Firstly synchronization is tested. The state, in which the node is ready to transmit to the server, is *SNDelta1.DeltaTransended*. At this point the clock should be exactly equal to $g_{MACcycleLength}$.

$$\begin{aligned} E <> SNDelta1.DeltaTranscended\ \&\& \\ gClk != gMACcycleLength\ (false) \end{aligned} \tag{2.6}$$

It should of cause not be possible for two nodes to accept the same message, therefore the following relation must be false:

$$\begin{aligned} E <> SNDelta1.NodeReceivedResponse \&\& \\ SNDelta2.NodeReceivedResponse\ (false) \end{aligned} \tag{2.7}$$

But two nodes are welcome to wait for a message at the same time:

$$\begin{aligned} E <> SNDelta1.BaseSSN \&\& \\ SNDelta2.BaseSSN\ (true) \end{aligned} \tag{2.8}$$

If only one channel is enabled in the base-station, the following relation will be false. But given that more than one channel is enabled, the base-station can have a message for both nodes, and then it will be true.

$$\begin{aligned} E <> SNDelta1.BaseSSN \&\& \\ SNDelta2.BaseSSN \&\& \\ gTotalOweResponse > 1\ (true) \end{aligned} \tag{2.9}$$

The following relation establish that when the base-station is responding, a node will be listening. This is true, but that is not the same as promising that the signal is sure to arrive.

$$\begin{aligned} A[]\,&BaseStation.Responding\,imply \\ &(SNDelta1.BaseSSN\,|| \\ &SNDelta2.BaseSSN\,|| \\ &SNDelta1.NodeReceivedResponse\,|| \\ &SNDelta2.NodeReceivedResponse)\ (true) \end{aligned} \tag{2.10}$$

In this chapter the relations sufficient to conclude the well-behavior of our model has been given. It is assured that:

- The System does not Deadlock
- The System exhibit Liveness
- The System does not exhibit Fairness
- The System follows the proposed protocol
- The System is synchronized
- The Nodes will not be in dis-consistent states
- The Possibility of nodes sharing a channel was analyzed.
- The Parameters have been analyzed and allowed relations have been given.

Therefore the listed relations must be sufficient to validate the model. All of the above relations have been tested, and found to yield the expected. It should also be noted that the above system was tested with two nodes. The result from using more nodes was an exploding state space which meant, that only a few of the invariants listed in this chapter could be proven. The available computational power was the SunFire server at *IMM, DTU* with twenty four UltraSparc-III/750 MHz processors and fifty-four gigabyte of available memory. Unfortunately *UppAal* was only capable of using a single processor at a time and limited the memory consumption to four gigabytes. An effort from the creators of *UppAal* to make use of distributed computation could yield a major reduction in testing time.

2.6 Delta vs. Time Protocol

In the article on "A new Protocol for Low Power Sensor Networks", the *Delta* protocol is compared to the *Time* based protocol. It is claimed that using the Delta protocol the node potentially will have to transmit fewer times, while obtaining the same resolution. This will now be proven true on a formal ground:

First we let the environment be interesting "allways". If this is the case it is found, that a maximum of 30 time units will pass from the environment is interesting and till the node is trying to transmit - both for the *Delta* and *Time* protocol. Thus, using the global variables listed in section 2.4.1, we find that the following relation is true for the value 30 and false for the value 31:

$$E <> Monitor.sending \&\& Monitor.lClk > 31 \; false \tag{2.11}$$

It is interesting to consider what happens to the relation a change from 5 to 200 happens in the global variable $g_{EnvironmentChange}$. From what was proposed in the original article, the *Time* node should send six times every time the *Delta* node sends a single message. The *Delta* timing indeed has changed, which can be seen by the following relation now evaluating as true:

$$E <> Monitor.sending \&\& Monitor.lClk > 180 \; true \tag{2.12}$$

Changing the $g_{EnvironmentChange}$ variable must have no influence on the *Time* protocol timing relation. This is proven by running equation 2.11 on the *Time* model again with the new global constant and obtaining the same result as before.

Observing this result, the mission of this chapter, to formally verify the original article propositions, is concluded successful.

2.7 Future Opportunities and Challenges

The next step in evolving this model would be to implement real node energy consumption values for the different node application actions. Also, measuring the time, in which the node is idle and requiring some power for this should be possible. With these feature implemented it would be possible to directly see the difference in energy consumption in a number of interesting cases. For instance it would be possible to establish in exactly which cases the *Delta* protocol is a better choice than the *Time* protocol.

Synchronization is a challenge which has yet to be inferred in a realistic manner to the model. This subject is not touched in the article, but after communicating with the authors it is clear that certain possibilities exist, and has been explored by the authors of the *A.N.P* protocol. In the future work with this model, it must be considered whether the energy required for synchronization is enough to have influence on the protocol performance.

2.8 Conclusion

In this report it has been shown that formal analysis is a unique and powerful way of evaluating and comparing models. It was shown that it is possible to model a complete network including base-station, nodes and the surrounding environment. It was possible to verify proposed protocol properties from the original *A.N.P* article. Also the limits of the network holistic simulation was met. With the available computational power it was only possible to verify small scenarios. Implementing energy consumption into the model will only make the simulation space larger and worsen this situation. As a consequence, formal verification of network holistic models is determined inappropriate with the currently available computational resources. In Chapter 3, a different approach to modelling is presented which yields better results in reducing the state space.

CHAPTER 3

The E.N.D Model

3.1 Introduction to The E.N.D Model

Upon deployment of a sensor network it is often critical to *E*stimate the time to *N*ode *D*eath (*E.N.D*).

Compared to the *A.N.P* model, presented in Chapter 2 the *E.N.D* model has focus on a single target node instead of an entire network. The surrounding environment of the node consisting of the neighbor nodes and the base station, is modelled through event streams. The result is a slim state space which can be verified and analyzed on a powerful workstation.

The protocols used in Wireless Sensor Networks do their best to limit overhearing other messages, and if possible never send a message which will not be received. Whether this is done using a clever scheme or using a separate communication channel, it is in this chapter shown, that the *E.N.D* model can help the designer analyze the worst case situation in a given scenario.

This chapter is structured as follows. First an introduction to the model and implementation details of the *E.N.D model* is given. Then an analysis of the model is performed using the *UppAal* framework and the developed environment for Testing and Analysis. The model is verified with the results obtained in several important articles on sensor networks. Finally, before the conclusion, an introduction to the *E.N.D* model generator is given.

3.2 Model

3.2.1 Interactions

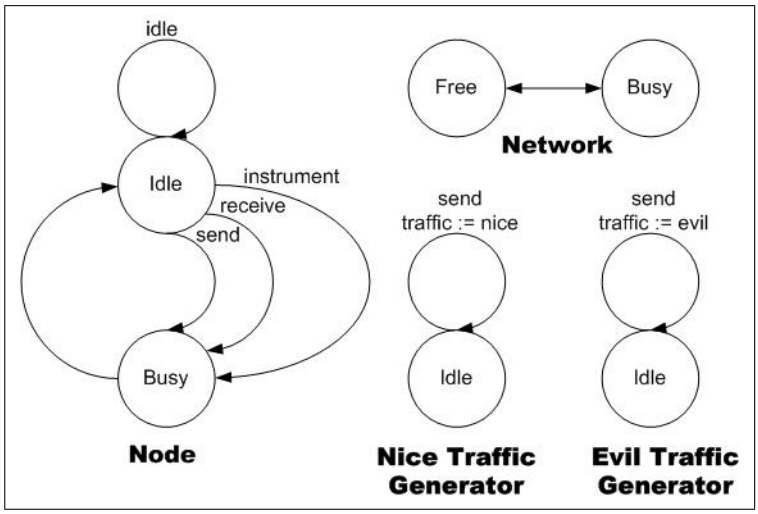


Figure 3.1 *The Model*

Node

The interaction between the node and the environment is modelled through events which will happen in a way similar to interrupts in micro controllers. In figure 3.1 it is demonstrated that the node should react to the *Send, Idle and Instrument* interrupt, along with the *Receive event*. Each of these actions will cause the node to be busy for a certain interval where after it returns to the idle state.

Nice Traffic Generator

Traffic which is meant to be received by the node is named as *Nice Traffic*. The nice traffic generator is normally in state idle and then in regular intervals create a *send* event which may be caught by the target node. The nice traffic generator model can be seen as a subpart of figure 3.1.

Evil Traffic Generator

Traffic which occupy the network around the node is named *Evil Traffic*. The evil traffic generators is normally in state idle but in regular intervals create a *send* event which may be caught by the target node. The generator of evil traffic model can be seen as a subpart of figure 3.1.

Network

The model of the network basically determines whether the electromagnetic waves around the target node are free or busy/occupied.

Base-station

In many protocols the base station does not initiate any transmission. A base-station model is therefore not required. A signal sent by the target node is sent as a broadcast, and therefore a receiving base-station is implicitly modelled. Should a base-station contact a node, this role is equal to a nice traffic generator.

3.2.2 Energy

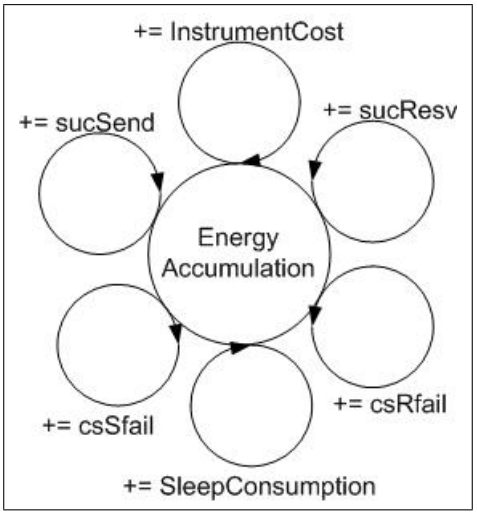


Figure 3.2 *The Energy Accumulation Model*

Energy is accumulated through actions. The details of the operating system, node implementation and topology is abstracted into a set of events which will happen in a certain time span or interval. For a given system it is necessary to measure the cost of entire actions: *"Completely Receiving a signal"*, *"Using the instrument"* and *"Completing a Send transmission"*. If using carrier sensing technology, the actions *"Determine the network as busy and postpone send action"* and *"Tried to receive signal but it was determined addressed for other node "* must also be measured. The energy model is visualized in figure 3.2. The meaning of the parameters are explained in the following listing:

- sucSend : The energy required for a send transmission
- sucResv : The energy required to receive a message
- csSfail : The energy required to check the network status before a send.
- csRfail : The energy required to check weather a message should be received or not.

- SleepConsumption : This is the energy usage when the system has been sleeping/idle for idleEnergyAccTime time units.
- InstrumentCost : The energy required to use the instrument.

With these energy consuming actions determined formal modelling will control the interplay and for example determine, how the worst set of interactions would occur.

3.2.3 Protocol

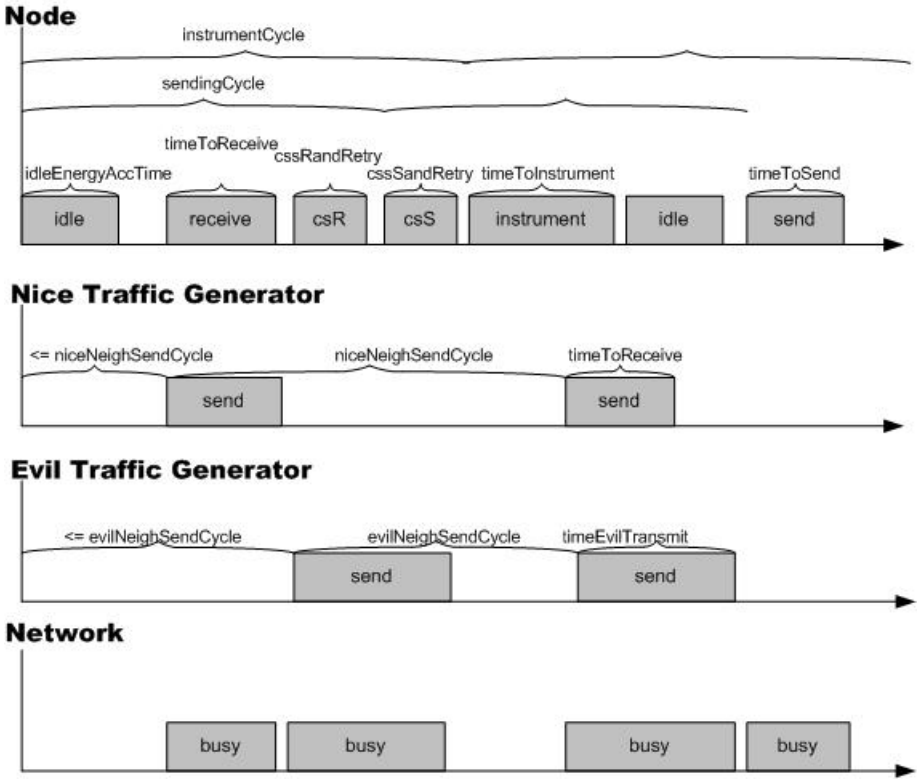


Figure 3.3 *An example showing the model parameters and interplay*

The *E.N.D* model aims at being generic in terms of protocol choice. This requires a range of variables necessary to define the properties of the used protocol. In figure 3.3 the protocol parameters is introduced and the interplay described. To give a natural feel for the parameters, figure 3.3 is now described with words:

1. Because nothing happened in the start of the test *idleEnergyAccTime* is passed threshold which describes the system energy consumption in idle/sleep state.
2. After an offset of less than *niceNeighSendCycle*, the nice event generator generates a send event. Since the network was free and the node idle, the node begins receiving the signal.
3. After an offset of less than *evilNeighSendCycle* the evil generator generates a send event. Since the node uses carrier sensing techonology, the node is only occupied for *cssSandRetry* time and then returns to idle state.
4. After *sendingCycle* time units, the target node itself will try to send a message. At his point the network is occupied, so the target node returns to idle state after *cssSandRetry* time units.
5. After *intrumentCycle* time units the node starts using it's instrument.
6. The node is now busy using the instrument and does not see that the *Nice* and *Evil* generator both generates a send event.
7. After using the instrument nothing happens for a while, so the node is sleeping in the idle state. This also causes a small energy consumption.
8. The Node "send" cycle has been reached and again it sends a message. The transmission takes *timeToSend* time units to complete.

With this basic feel for the inter-event-play a thorough introduction to the introduced parameters is now given.

- sendingCycle : This is the sending cycle of the target node.
- instrumentCycle : If the instrument is enabled, it is used with this interval.
- idleEnergyAccTime : Since the *UppAal* frame cannot do math on clocks (only assign values), it is necessary to model the energy consumption, when sleeping, in discrete steps. This value determines the minimum time the system should be idle/sleeping to consume an amount of energy.
- csSandRetry : If the system tries to send a message but determines that another node is using the available bandwidth, then the node waits this amount of time before trying again. For synchronous systems this value is equal to *int x sendingCycle*. Some protocols such as the *S-MAC* will use a random back off period. This should be implemented in a future version of *E.N.D*.
- cssRandRetry : If the system is receiving a message but determines that the bandwidth is occupied, it will take this amount of time before the system has returned to the idle state. In the next model version "retry" should be deleted from the variable name.
- timeToSend : This determines the time to send a message.
- timeToResceive : The time to receive a message.
- timeEvilTransmit : The length of evil messages.
- timeToInstrument : The time required to use an instrument attached to the node as explained in section 1.1.1.

- niceNeighSendCycle : The send cycle of the nice neighbors.
- evilNeighSendCycle : The send cycle of the evil neighbors.

With the model defined, the next step is to create a timed automata from the current automata. This will be explained in the next section on the *UppAal* implementation.

3.3 Implementation

The global variables and process assignments define and shape the model. The goal is that designers working with the *E.N.D* model should be able to model their system from these variables alone. Therefore understanding each of these variables is vital. More than half of the variables were presented in the chapter on the model, and the last are now defined:

- niceNeighbors : Boolean value, determines if the nice neighbors are active.
- evilNeighbors : Boolean value, determines if the evil neighbors are active.
- Instrument : Boolean, determines if the instrument is used.
- cs : Boolean, determines if carrier sensing technology is used.
- pMax : The maximum energy constant. It has to be larger than the energy consumption.
- maxM : The maximum messages constant. It has to be larger than the maximum number of events such as send/receive/instrument use.
- testTime : The amount of time units for which the test is run.
- block : The maximum number of times which the system will block an event.

The process and system assignments are important when the designer wish to have more than a single nice and evil neighbor node. Nice node neighbors are defined as nodes which communicate with the target node. Evil nodes are defined as generators of passing traffic, thus interfering with the network around the target node. The following listing is an example of a system consisting of: the node for analysis (target node), one nice neighbor, one evil neighbor, a network and finally a force entity necessary for creating urgent edges in *UppAal*.

Process and System Assignments

```
1 <system> system node,nneigbor,eneighbor,network,force;</system>
```

The constants and process assignments which have been introduced, should make the following description of the individual models straightforward.

3.3.1 The Node

The node is built around a single state in which time may pass and a set of actions performed upon an interrupt or event. The node model state is therefore largely defined through variables and clocks rather than places.

The main state is called *idle*. From this state events from clocks or concurrently running model will trigger the node to do one of the following actions:

- Send a message
- Receive a message
- Use the instrument
- Accumulate idle energy
- Change the busy-flag

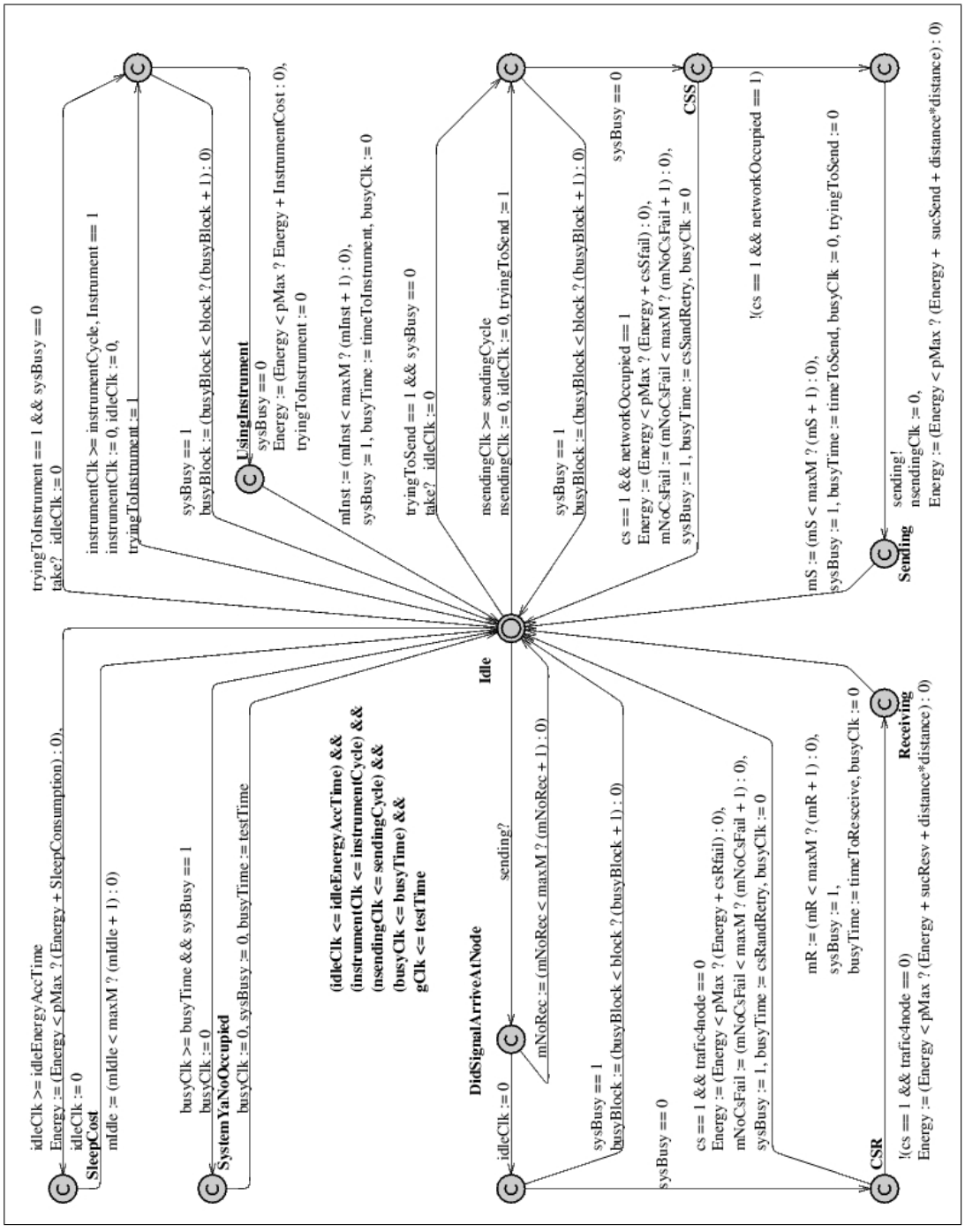


Figure 3.4 *The Node*

As can be observed in figure 3.3.1 the model is complex and a close analysis is

necessary. The graphics should be understood as follows:

- The places are displayed as circles. Time cannot pass in places marked with a *C*. The starting place has an internal circle.
- Transitions are arrows. Each transition may hold one or more from the following data set: Invariant, Synchronization, Variable assignments. In the model they are continuously displayed in that exact order, next to the transition. The invariant must be satisfied before the transition is enabled.

Send a message

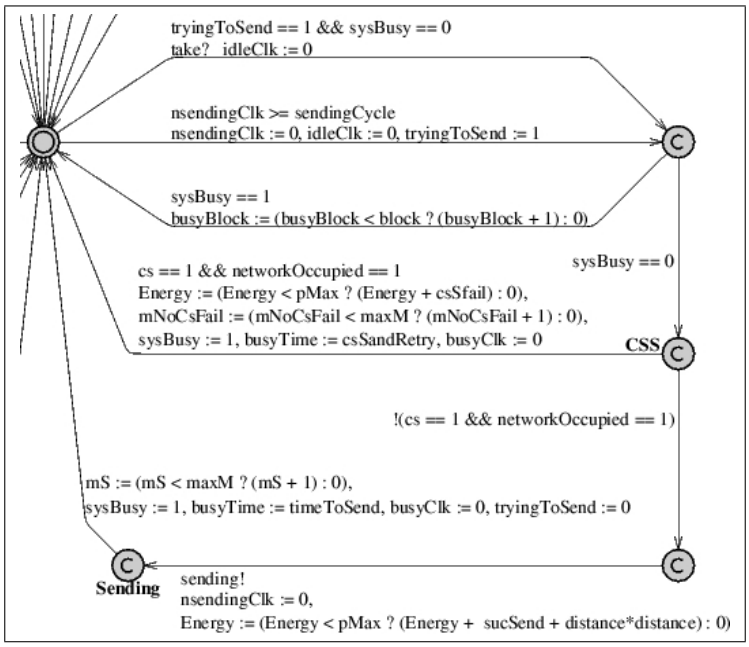


Figure 3.5 *Sending a Message*

The action of sending a message can be seen in the model in figure 3.5. Analyzed from the top and down the first transition does the following: If the node is trying to send and it should do so as soon as the system is not busy, also at the same time the idleClk is reset so the discrete sleep interval is disrupted. The next transition which will start a transmission is when the clock *nsendingClk* is equal to the *sendingCycle*. When this happens the *nsendingClk* and *idleClk* are both reset, and the variable *tryingToSent* is set. Once the node is trying to send the next transmission will cause it to fail if the system was busy upon the send event. If this happens the *busyBlock* variable is increased. This will allow for analysis of how busy the node has been. If the node is not busy the model will reach the *Carrier Sensing State* (*CSS*). If carrier sensing technology is enabled and the network is occupied the node will pros pone the send action. Otherwise it will perform the send. Upon either action the energy is accumulate as expected and also the

variables *mNoCsFail and mS* used for analysis. An important set of variable assignments is:

```
Setting the node in busy mode
sysBusy := 1, busyTime := csSandRetry, busyClk := 0
```

An important thing to notice is what will happen if the network is determined occupied and the signal then resent. In several articles such as *S-MAC* they use a random back off time span and they tries again. In the current *E.N.D* model it will be tried to send again after the fixed time *csSandRetry*. In a future *E.N.D* model random back off can be implemented. This will have a major impact on the state space though.

Receive a message

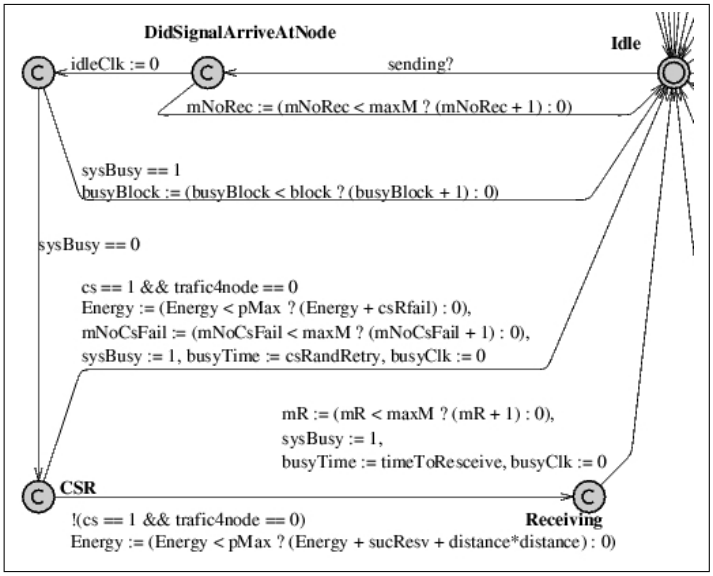


Figure 3.6 *Receiving a Message*

The model subpart responsible for message is shown in figures 3.6. The model leaves the idle state in the upper right corner upon synchronizing with over the *sending* channel with a neighbor. The mode is then free to decide if the signal arrives at the node or not. If the signal arrives it is checked if the system is currently busy or not. If the system is busy the *busyBlock* variable is increased and the signal is not received. If the system is not busy the *CSR* state is reached. In this state it is determined if the node is capable of differentiating signal directed at the node or not. If so the nodes quickly abandons the malicious signal, otherwise the signal is received and the appropriate variables increased.

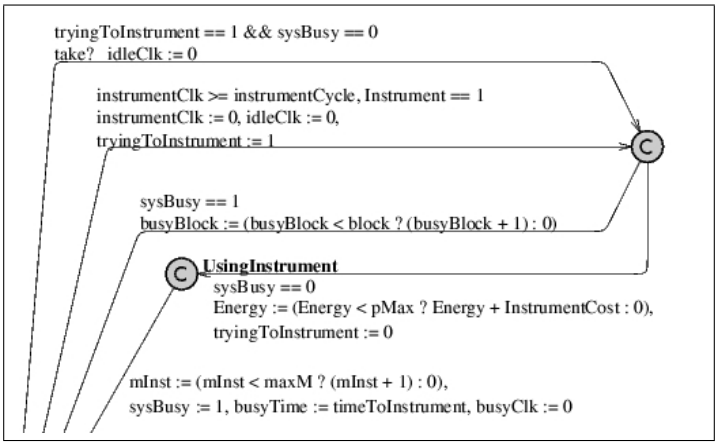


Figure 3.7 *Instrument Usage*

Use the instrument

In figure 3.7 the part of the node model responsible for handling instrument usage is shown. There are two ways the use of the instrument may be initiated, this is the two top most transitions. The basic way is if the instrument cycle threshold has been passed an the interrupt has been sent. If the system is busy at that exact time the topmost transition will make sure the instrument is used as soon as the node is again idle. The third transition from the top is taken when the system is occupied. With the pre-control surpassed the instrument is used in the transition closest to the bottom, in this case appropriate variables are increased and set.

Accumulate idle energy

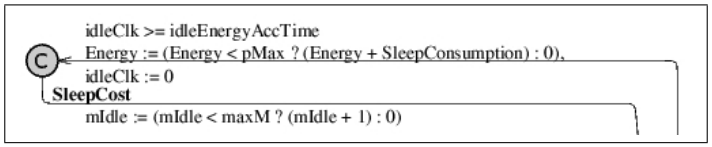


Figure 3.8 *Idle Energy Accumulation*

The preferred way of modelling the energy dissipation due to the node sleeping would be *SleepingClock * IdleEnergy*. Unfortunately the SleepingClock in *UppAal* is a span of time rather than a counter, which prevents doing this kind of mathematic operation. Therefore the idle energy must be increased in fixed discrete steps. For example the designer defines the smallest time in which 1 energy unit is dissipated when sleeping. The measure will not be perfect, but the approximation will often be acceptable, as is later shown in section 5.3 on model verification. This part of the node model can be observed in figure 3.8. For this way of modelling it is required that all other actions

resets the *idleClk*.

Radio Distance

The following five actions are determined to dissipate energy in a node: Idle, Sending, Receiving and Instrument usage consumption. Finally the direct influence of the transmit distance can be model. This is done as follows. The node is instantiated with a distance parameter which is then used in the energy usage accumulation used when receiving or sending a message. As explained in article [12], the signal power received at a certain distance decreases with the distance d as follows:

$$P_r \alpha P_t d^{\beta} \tag{3.1}$$

where beta is between two and five. The *UppAal* framework does not define a mathematical operator for raising a variable to a certain power, but assuming a beta value of two (free space) means that the required transmitted power and thus also energy, is proportional to *distance*distance*, which *UppAal* allows. With a stronger mathematical model in *UppAal* a typical β value of 3.8 could be used. Many scenarios will model the energy consumption better through directly using the *send and receive* variables, but the distance feature allows for a direct simulation of transmission distance.

The Idle State

The invariant in the idle state is as follows:

$$\begin{aligned}(idleClk <= idleEnergyAccTime)\,\&\&\,(instrumentClk <= instrumentCycle)\,\&\&\\ (nsendingClk <= sendingCycle)\,\&\&\,(busyClk <= busyTime)\,\&\&\,(gClk <= testTime)\end{aligned} \tag{3.2}$$

The first four sub parts of the invariant makes sure to force the node to take certain transitions at or after a certain time. This is a necessary and common design pattern in *UppAal* models. A new and not intuitive invariant is the last part reading: gClk <= testTime. In the world of protocol verification, invariants like this would never be seen. For energy accumulation verification, it is a very nice invariant since it will stop all actions happening after the gClk has passed the testTime threshold. When searching for an answer to an equation like:

$$E <> Time < X \,\&\&\, Energy > Y \tag{3.3}$$

UppAal would continue to search for possibilities even after the Time clock had passed Y, unaware that the clock would never be reset. Inferring this hard deadlock makes sure no unnecessary state space is generated.

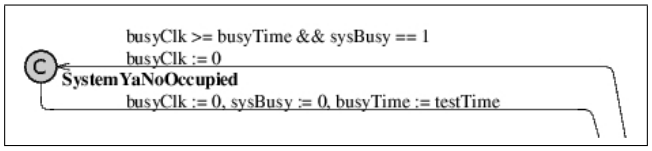


Figure 3.9 *Busy-Flag*

Change the busy-flag

The system "busy" flag is controlled through the sub model in figure 3.9. The busy flag and busy time is set every time an action is performed as elaborated in the section on the *send* action. This part of the model ensures that the node will not perform anything before the busy time expires and the model again can perform actions.

Summary

The node is modelled as a single non committed state connected with urgent transitions and committed states. Thus time may only pass in the state *idle*. The system state is thus defined through variables such as *busy, busyTime, Energy, etc*.

3.3.2 Nice Neighbors

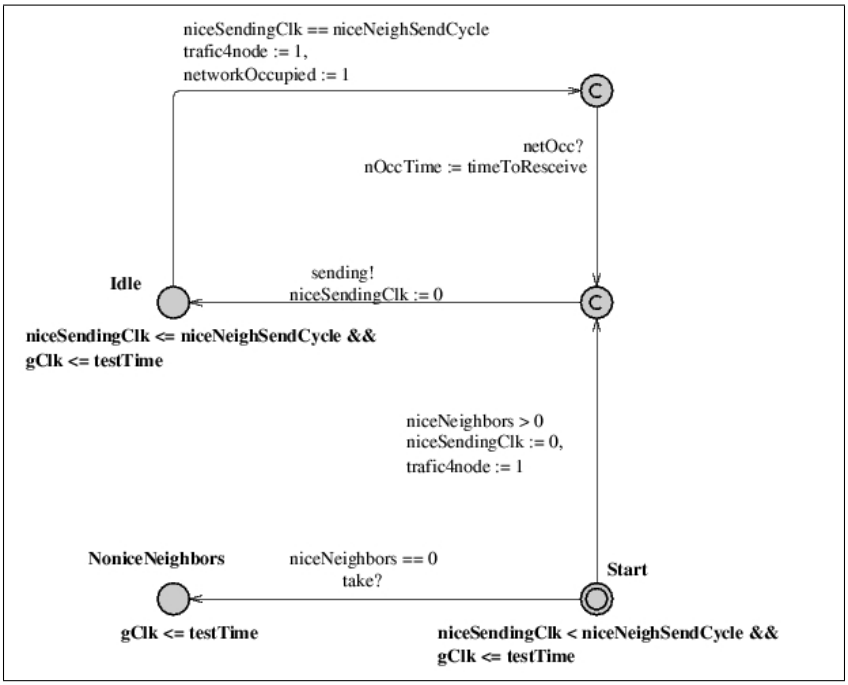


Figure 3.10 *Nice Neighbor*

The nice neighbors generate a stream of message events addressed for the node targeted for analysis. It is a synchronous event stream, but the starting point may be varied freely by the model between zero and the *Nice Node* send cycle. The node contains an extra state: *NoNiceNeighbors* where it resides when the designer is creating a topology without nice neighbors. Upon simulation startup without nice neighbors the transition leading to this state will be enabled, and the urgent force synchronization will ensure that the nice node will be locked for the remaining simulation run. When analyzing the trace this yields a faster general view of the model state. The nice neighbor can observed as figure 3.10. Beside the inter node functionality a send cycle also communicates with the network model. The design pattern used is the same as the *BusyFlag* in the node model. When a neighbor transmit a message it stores the time it will set the shared variable *nOccTime* and then synchronizes with the network. The network will respond by being busy the next *nOccTime* time units.

3.3.3 Evil Neighbors

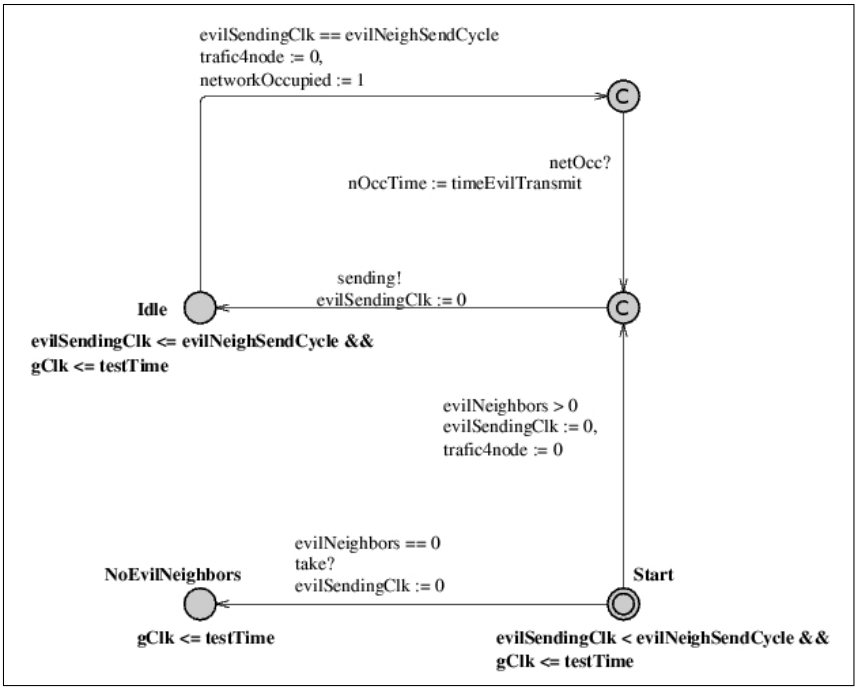


Figure 3.11 *Evil Neighbor*

The evil neighbors generate a stream of message events passing the node targeted for analysis. The evil neighbor model is shown in figure 3.11. It is a synchronous event stream, but the starting point may be varied freely by the model between zero and the *Evil Node* send cycle. As the *Nice Neighbor* the *Evil Neighbor* features a dead state *"NoEvilNeighbors"* used, if no evil neighbors are chosen active by the designer.

The synchronization with the *Network* model is the same as the *Nice* neighbor.

3.3.4 Network

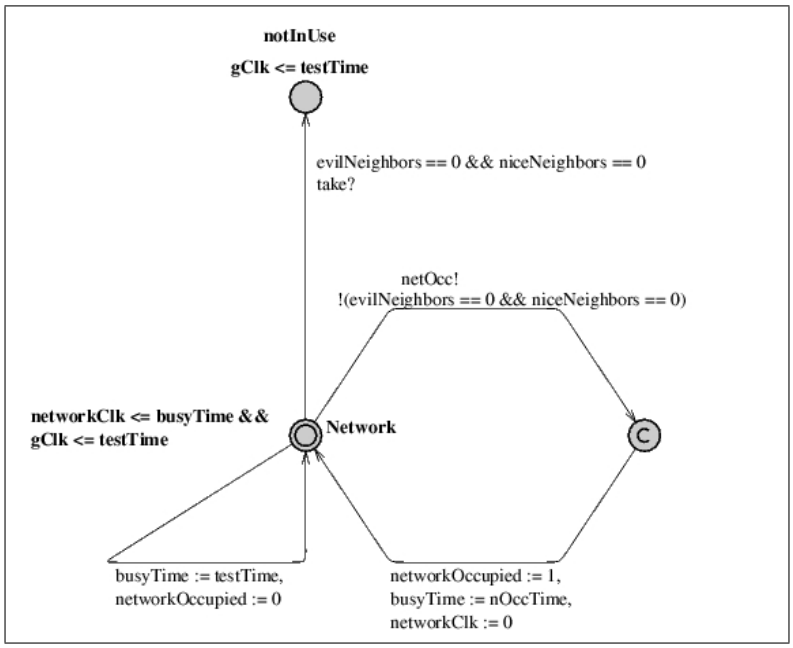


Figure 3.12 *The Network*

The Network model defines whether the network is busy or free for transmissions. The network model is demonstrated in figure 3.12. This information is necessary when dealing with carrier sensing technology. The network is initiated in the *Network* state. The transition to the right is taken upon a synchronization with either a *Nice* or *Evil* neighbor. After this synchronization the network will be occupied in an interval determined by the synchronizing model. The transition below the *Network* state is taken when the *Network* should no longer be occupied. As the *Nice Neighbors* and *Evil Neighbors*, the node features a "dead" state. This is reached using the topmost transition and only used if no nice or evil neighbors are chosen active by the designer.

3.3.5 Force

The *Force* model in figure 3.13 is necessary to create urgent transitions. Urgent transitions are used when a model should take a transition as soon as it becomes enabled without time passing. The *UppAal* framework does not have a syntax for describing this transition type, but instead it is common to use this design pattern to solve the issue.

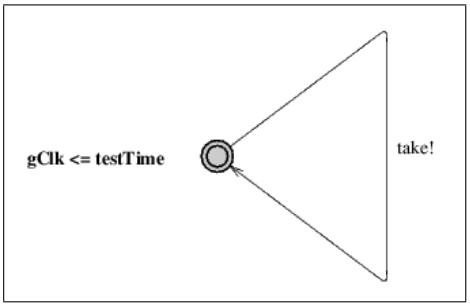


Figure 3.13 *The Force Model*

3.4 Analysis

In this section the results from exploring the safety, liveness and interesting properties of the model are presented.

3.4.1 Safety

In this subsection the model response to normal safety invariants are examined. First it is checked if the model will ever deadlock.

$$A[]\, not\, deadlock\ (false) \tag{3.4}$$

From invariant 3.4 it is clear that the system will indeed deadlock. As already explained in chapter 3.3 on the implementation this is implemented deliberately, in an attempt to minimize the state space as much as possible. In fact, the system should always deadlock after the testing time has expired. This is proven through invariant 3.5.

$$A[]\, gClk > testTime\, imply\, deadlock\ (true) \tag{3.5}$$

As expected the system will be deadlocked as soon as the test time has passed.
The following invariant 3.6 proves that the network will always be occupied while either an evil or nice node is transmitting.

$$A[]\ (nnb.Sending\ or\ enb.Sending)\ imply\ networkOccupied == 1\ (true) \tag{3.6}$$

Therefore the node targeted for analysis may securely use this information

3.4.2 Energy

There are two corner cases regarding energy consumption which can be checked through this way of modelling: the lowest possible energy consumption and the highest possible energy consumption in an interval.

Lowest Possible Energy Consumption

Determining the lowest possible energy consumption can be done through finding the value where the result of invariant 3.7 change from *true* to *false*

$$\begin{aligned} &A[]\ gClk > 500\ imply\ n.Energy > 15\ (true) \\ &A[]\ gClk > 500\ imply\ n.Energy > 16\ (false) \end{aligned} \tag{3.7}$$

It can thus be determined that the lowest possible energy consumption for the analyzed node is 15 energy units. This method of analysis yields some potential for discovering new optimal ways of node behavior.

Highest Possible Energy Consumption

Finding the highest possible energy consumption for a system has great potential. Whether caused by nearby earthquakes, hungry animals or silly protocols it is ignorant to design a system without concern for the worst case scenarios.

$$\begin{aligned} E <> gClk < 1000 \;\&\&\; n.Energy > 131 \;(true) \\ E <> gClk < 1000 \;\&\&\; n.Energy > 132 \;(false) \end{aligned} \tag{3.8}$$

Invariant 3.8 show that the maximum consumed energy in a time interval for the specified scenario indeed can be found. Finding the values 131 and 132 is a matter of performing a manual binary search. This is very time consuming and a tool for performing this analysis automatically should be developed.

3.4.3 Interesting Properties

Several other interesting properties can be verified formally. The first scenario is about using carrier sensing (CS) / adaptive sleeping technology. Invariant 3.9 will be false when using CS while it will be true when not using CS technology.

$$E <> n.Sending \&\& networkOccupied == 1 \;(false) \tag{3.9}$$

The immediate result is that not using CS technology exposes the system to the a where it is sending or receiving messages in vain because the messages are disrupted by a busy network.

The next invariant 3.10 is an unfortunate consequence of a wireless system.

$$nnb.Sending -- > n.CSR \;(false) \tag{3.10}$$

It is shown that the event of a nice neighbor sending a signal will not always cause the main node to actually discover this. The signals may not arrive at the node, or the node may be busy at that time.

$$\begin{aligned} A <> n.CSS \;(true) \\ A <> n.CSR \;(false) \end{aligned} \tag{3.11}$$

Finally it is shown through invariant 3.11 that the node targeted for analysis always will try to send, which happens in state *CSS* while even though the neighbor nodes are trying to send signals they may never actually arrive. The second part of the result is thus a result of what was already found in 3.10.

3.4.4 Summary

In this section a range of vital model properties was formally verified. It was also shown that the *E.N.D* model can be used for finding the trace resulting in the highest energy consumption. This is very valuable for designers wanting to take counter measures in future designs.

3.5 Conclusion

In this chapter the generic *E.N.D* model of a sensor network was implemented. The *E.N.D* model distinguishes it self from the *A.N.P* model from chapter two because it is centered around a target node and the surrounding network is insinuated through event streams, which is much faster then simulating the complete network. After the implementation of the model it was analyzed using formal verification. This yielded the expected behavior and basis for trust in the model in future tests. It was also discovered that the manual binary search necessary to determine ranges of variables were extremely time consuming and that the designer/tester should be relieved from this burden.

Because the *E.N.D* model is generic it is possible to simulate different protocols, systems and scenarios. This will be examined in chapter 5 on *Test and Results*. Changing and/or testing the *E.N.D* model by hand is both time consuming and potentially confusing. Therefore a *Design and Analysis* framework for the *E.N.D* model has been developed and will be introduced in chapter 4.

CHAPTER 4

The E.N.D Design and Analysis Framework

4.1 Introduction

The *E.N.D* framework is enables the following features for the user: A GUI for design and creation of *E.N.D* models, *Explore* scripts which find the *scenario worst case* energy consumption and an analysis script for automatic analysis of traces. Figure 1.10 demonstrates the interactions between the *E.N.D* framework components and *UppAal*.

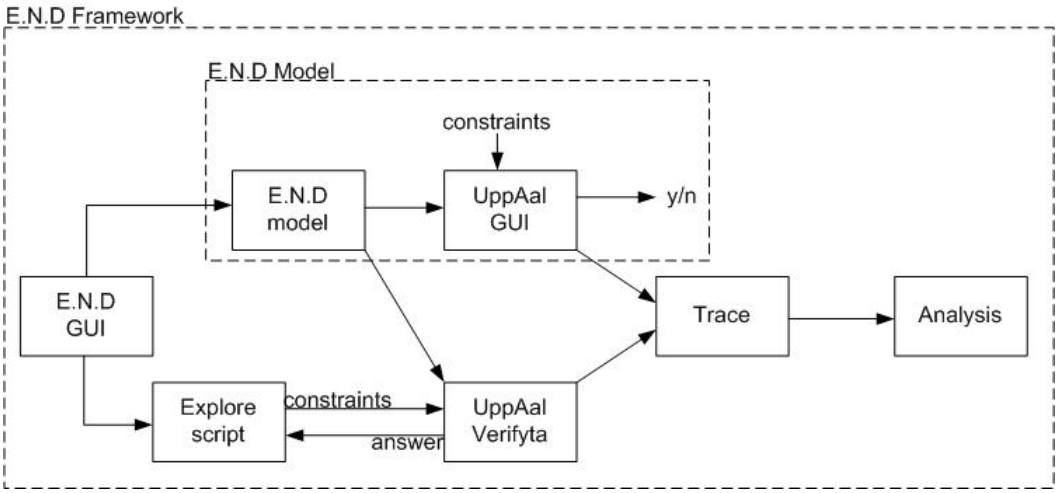


Figure 4.1 *Analysis and Verification of sensor networks*

Figure 1.10 is divided into two part using dotted boxes. The smallest dotted box at the top is the normal *UppAal GUI* interacting with a manually edited *E.N.D* model. The *UppAal GUI* allows for manual analysis of constraints with a *satisfied or not satisfied* (y/n) answer. This type of analysis was performed in section 3.4.

The *E.N.D* framework extends this basic analysis. Explained from left to right in figure 1.10 the component purposes is the following:

- The *E.N.D GUI* is used to create *E.N.D* models and *Explore* scripts.
- The *Explore* script interfaces the *UppAal* tool *verifyta* used to verify timed automata using a command line interface.
- The generated trace can be analyzed using the *Analysis* script.

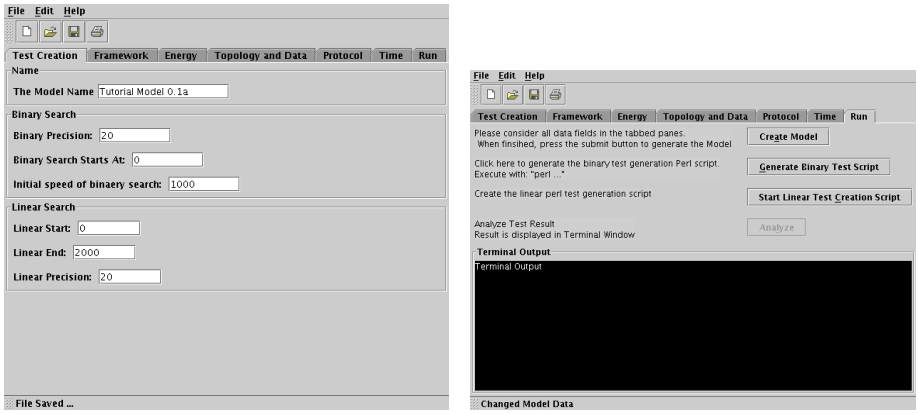
In this chapter an introduction to each of these tools is given.

4.2 E.N.D GUI

The *E.N.D GUI* enables the designer of sensor nodes new possibilities to systematically test interesting scenarios using an attractive graphical interface. The program features include:

- Load / Save / Print of model Data
- *UppAal* model Creation
- Explore Script Generation
- Integrated Test Environment (not finished)

The application was developed under Linux using the *Eclipse IDE*. The application allows the designer to work with the model meta data and control it very efficiently. The model properties are divided into intuitive groups which makes it easy to form a general view of the current model state and scenario. To get a feel for the program two screen shots of the program can be seen in figure 4.2 while a complete user guide is attached in appendix A.



(a) Explore Script Configuration (b) Model and Script Generation

Figure 4.2 *The E.N.D GUI*

The *E.N.D GUI* application can be found on the thesis CD in the directory /END. It can be initiated using the following command:

Initiating the E.N.D GUI

```
1 user@machine (CD/END) > java END
```

4.3 Explore scripts

An important discovery from testing the *A.N.P* and *E.N.D* models was the need for an automated test and analysis environment. The reason for this was the inefficient manual binary search needed to find the *worst case* energy consumption in a certain scenario. In this chapter it is explained why an automatic linear search is found to be faster than a binary search because of state space reuse.

4.3.1 Exploring

The goal of the explore scripts is to find the trace which yields the highest consumed energy within a certain time frame. This is done by asking a question such as:

$$E <> Clock < X \&\& Energy > Y \tag{4.1}$$

Which translates into *"Will there ever exist a state where the clock is less than X and the energy consumed is higher than Y"*. The UppAal framework is able to answer this question with *Satisfied or not Satisfied*. If the answer is satisfied, then a trace to the state is given. The problem can then be divided into two.

Verification

If the designer have entered the exact values for his system and know the limit of his battery in the chosen test time, he can read the answer as *not Satisfied: Yes your system will always have enough energy*, or *Satisfied: No, the system may consume more energy*. This basic case is sufficient for verification purposes, but insufficient for analysis purposes since little information is available on the worst case energy consumption. It is only known that it is less that Y in formula 4.1.

Analysis

Another approach to the system would be a wish to find what causes the worst-case situation to happen and the worst case trace. In this case the problem is to find the largest Y value in equation 4.1, for which the answer is *satisfied*. The trace file generated by UppAal can then tell the designer exactly how the worst case happened. This approach with the current UppAal framework requires a manual binary search to be performed, and thus can be very time consuming including frequent attention by the tester. This is extremely time inefficient, so development of an automated test tool was needed. In the following subsections two different automated test tools are presented. The first uses a customizable binary search and the latter a linear search which reuses the already generated state space. The latter is found often to be the fastest.

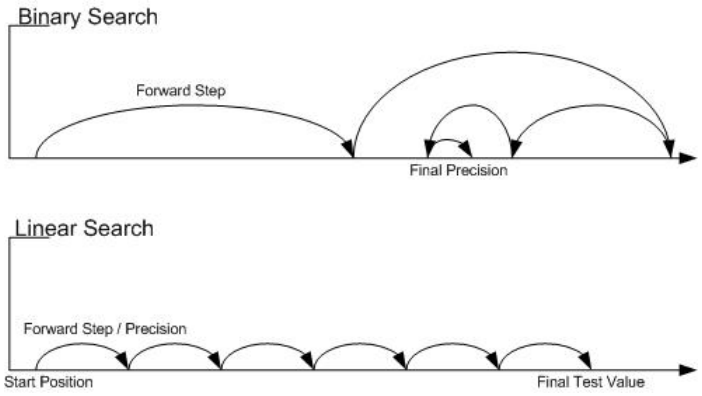


Figure 4.3 *Search Patterns*

4.3.2 Binary Explore Script

The binary search is defined through three constants.

- Start position
- Forward Step
- Final Precision

An example of a binary search is illustrated in figure 4.3. The binary explore script developed to care take this process and standardize the position of the resulting traces can be found on the attached cd in the directory: */testScripts/binary/binaryTestScript.pl*. Using this test tool is a matter of editing the constants and then initiating the search script with appropriate arguments which are explained if the script is executed without arguments. The script can also be generated from within the *E.N.D GUI*
Compared with the Linear Explore Script the Binary Explore script has many advantages. For example the test will stop as soon as the desired precision is obtained. Also uses the lowest amount of queries to find the result. Furthermore the tester is able to follow the progress of the search. The major disadvantage is that for every sub-result the state space graph is rebuilt completely from scratch which is very time consuming.

4.3.3 Linear Explore Script

The linear search is defined through three constant:

- Start Position
- Forward Step
- Final Value

An example of a linear search is illustrated in figure 4.3. The clever thing about the linear search is that it allows for state space reuse because the queries are known in advance. The script to create the queries can be found on the attach CD in the directory: */testScripts/linear/createQuery.pl*. Executing this file will create a file called magicQuery.q which can then be used directly with the UppAal *VerifyTA* tool like this:

```
Initiating a linear search
```

```
user@machine (/uppaal-3.4.7/bin-Linux) > verifyta -d -S0 -T -f
                        traces/TRACE -t0 model.xml magicQuery.q
```

The major disadvantage is that the search continues until the queries has been answered. It should be noted that the tester can manually check and stop (*ctrl-c*) this process as soon as the answers are *not satisfied*. Since the executions happens within the *UppAal* framework this issue has not been solved.

4.3.4 Comparing the Binary and Linear Explore Script

The pros and cons of *binary* and *linear* search have been explained in the two former subsections. To summarize, the major advantage of the binary search is that it uses the lowest amount of queries to find the result and tester is able to follow the progress of the search. This should be compared to the potentially faster linear search because of state space reuse. Both methods have disadvantages and thus the method of test is left to the designers preference. The issue have been discussed with the designers of the *UppAal* framework and allowing for state space reuse in a binary search will be a future feature, while it is currently known as a part of my *Enhancement Bug 120*.

4.4 The E.N.D Analyze Tool

The result of running either of the explore scripts is a trace file. This file can be loaded into the *UppAal GUI* with the model and the full trace followed in the simulation environment. An example of this can bee seen in figure 4.4. While this feature truly is very nice, it becomes very inefficient for larger with test scenarios. The *UppAal* framework accommodate this through the *libutab* parser library. The library allows developers to write custom trace analysis tools or interface the *tracer* utility which prints the trace to *stdout*. The latter solution was chosen because it complied fully to the *E.N.D* analysis requirements.

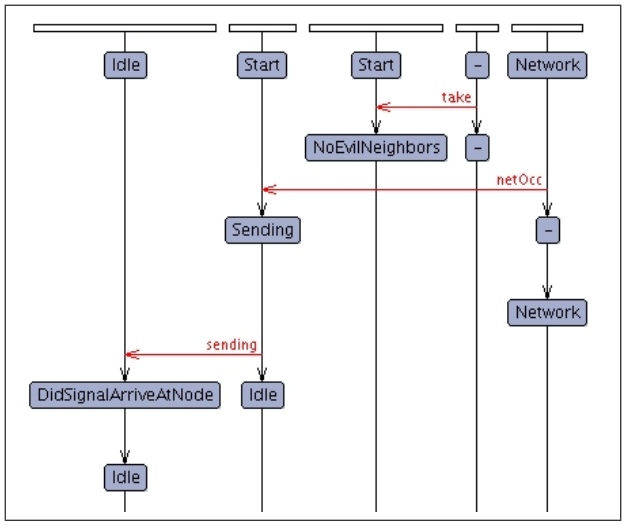


Figure 4.4 *Sequence Diagram Trace*

The Analyze Script

The */analyzeScript/analyzeFile.pl* script interfaces the tracer utility. It does so by having the *tracer* utility output the trace to stdout and then redirecting this to a file. Through regular expression the key variables in the final state is then fetched and appended in a chosen file, in the *results* directory. It is thus possible to gather test results with a basic relationship in a single file. An example of this can be seen in the following listing:

An example of a result file

```
1 ------------------------------------------------------
2 - Result: Topology: two evil, one nice, sendcycle 799 -
3 ------------------------------------------------------
4 n.Energy = 156
5 n.mS = 1
```

```
n.mR = 4
n.mIdle = 6
n.mInst = 0
n.mNoRec = 1
n.mNoCsFail = 0
n.busyBlock = 0
TestTime = 29

-------------------------------------------------------
- Result: Topology: two evil, one nice, sendcycle 799 -
-------------------------------------------------------
n.Energy = 156
n.mS = 1
n.mR = 4
n.mIdle = 6
n.mInst = 0
n.mNoRec = 1
n.mNoCsFail = 0
n.busyBlock = 0
TestTime = 29
```

In the example above two tests have been run and stored in the same file. The data can then be presented in the most relevant manner as is tried in chapter 5.3 on Model Verification. In the user guide attach in appendix A an introduction to using the *Analysis Script* can be found.

4.5 Conclusion

In this chapter the *E.N.D* framework has been presented. The three major components of the framework were presented as the following: *E.N.D GUI*, *Explore* scripts and *Analysis* script. It was shown that the *E.N.D GUI* is capable of generating *E.N.D* models and *Explore* scripts. The use and features of the analysis script was also presented. The *E.N.D* framework is thus a complete tool-set for analysis of sensor networks based on the *E.N.D* model.

CHAPTER 5

Tests and Results

5.1 Introduction

In this chapter tests and results are presented. First the tests are covered. This deals with creating a complex scenarios based on the *E.N.D* model. In each scenario one or more parameters are then changed and the result analyzed using the *E.N.D* framework presented in chapter 4. In the second part of this chapter the results achieved using the *E.N.D* model are presented. A successful result is obtained when it is possible to verify a known protocol property or proposed idea using the *E.N.D* model. In the section the model successfully verifies properties from four important articles in the area of sensor networks.

5.2 Tests

In this chapter the test strategy is explained. Then the test details and how to read the results are presented. Finally the tests are presented and the immediate results discussed.

5.2.1 Test strategy

An important factor in finding the worst case energy consumption is the number of receptions required by each node. This is best exemplified through three different topologies yielding very different reception histories.

- Tree, Inner position
- Tree, Outer position
- Line

To minimize the node energy consumption researchers propose different protocols. The main factors which they change in protocols are:

- Sending frequency
- Instrument usage frequency
- Carrier Sensing technology
- Message length

Hence these three cases are tested. Finally passing or "evil" traffic should be analyzed.

- Passing Traffic

5.2.2 Testing Explained

Each test will find the highest possible consumption of energy under the given circumstances.

The energy required to perform an action is given relatively to a similar test scenario and not in Joule. In this test we use the energy relations found in article [19] because the data are found to be the most precise data published. The values are:

1. Energy Consumption for Sending: 50
2. Energy Consumption for Reception: 20
3. Carrier Sense Send Fail: 10
4. Carrier Sense Receive Fail: 10
5. Instrument Energy Consumption: 5
6. Energy Consumption when Sleeping: 1

All tests are run for 1000 time units, while the event frequency and message length is varied from test to test. Finally the precision in energy consumption is decided to be maximum 1 energy units in the line graphs and 10 units in the bar graphs.

5.2.3 Varying Nice Node Sending Frequency

The first test on the system is changing the sending cycle of a nice node placed in a normal line topology. Surprisingly many facts about sensor network nodes establised from this test. The sending cycle is changed from every 50 time units to every 900 time unit. The evolution of the most interesting variables are shown in figure 5.1 while development of all variable values is placed in appendix B.3. The first important ob-

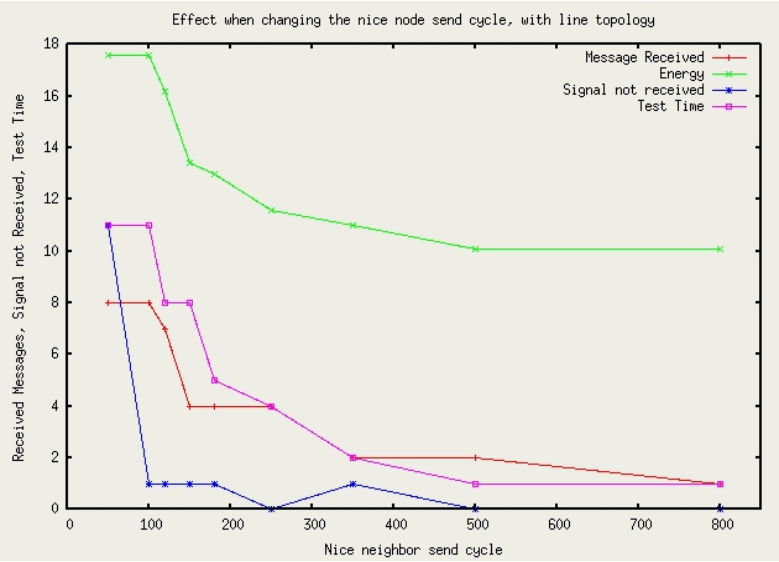
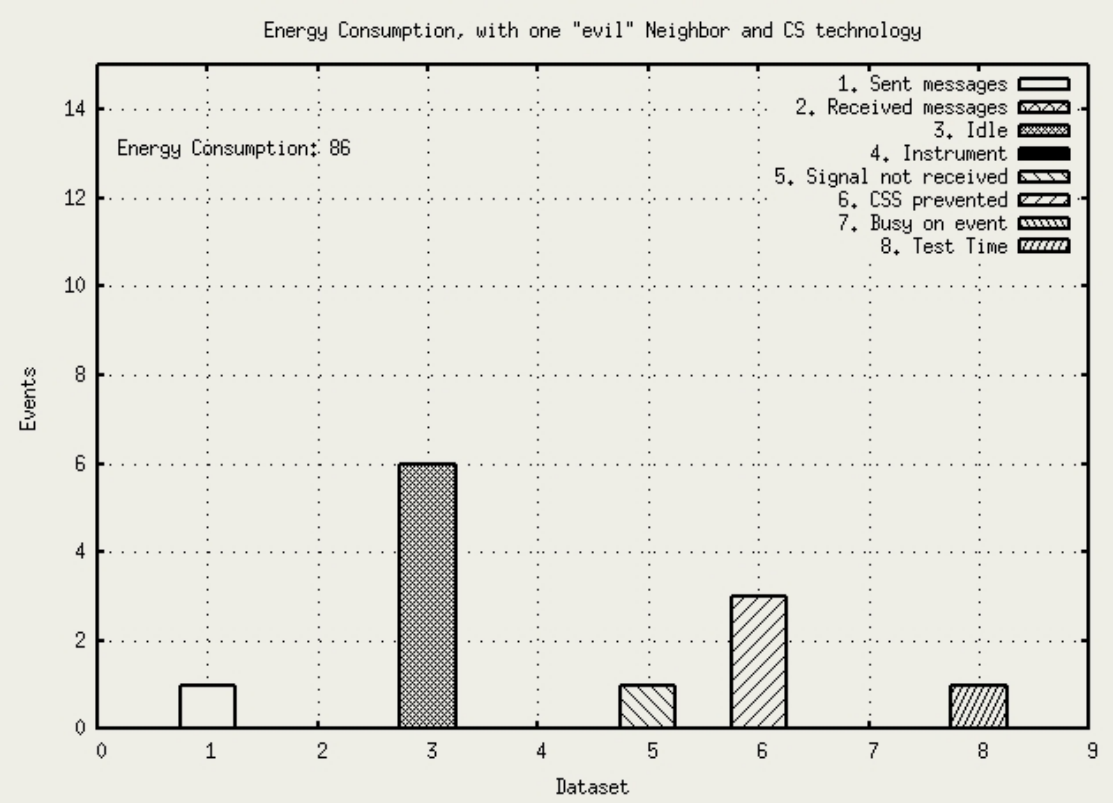


Figure 5.1 *Effect on received messages, energy, messages not received and test time, when changing nice node cycle*

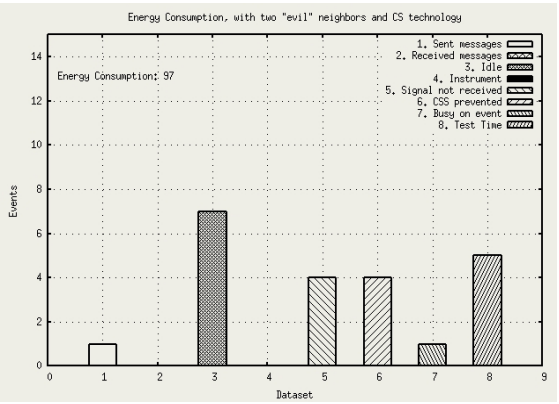
servations is done when the send cycle changes from 50 to 100. At 50 time units, the target node has reached the saturation and cannot receive more messages. The highest energy consumption will thus happen if 11 messages are successfully received and 11 signals are simply not discovered. At 100 time units the node will experience the same signals and use just as much energy, while the system behaves a lot differently trying to send only half amount of messages. As fewer messages are received it is seen that the energy consumptions is falling equally while stabilizing around 12 energy units, due to the fixed target node send cycle. From this test we can thus conclude that saturation can occur as expected and the worst case actually happens if many of the signals are not received. This latter conclusion is not what would be perceived by intuition. The situation is due to that fact that the system uses more energy to send a message then to receive a message. Thus for the system to send all possible messages, many of the signals from the neighbor node should not be received.

5.2.4 Carrier Sensing Technology

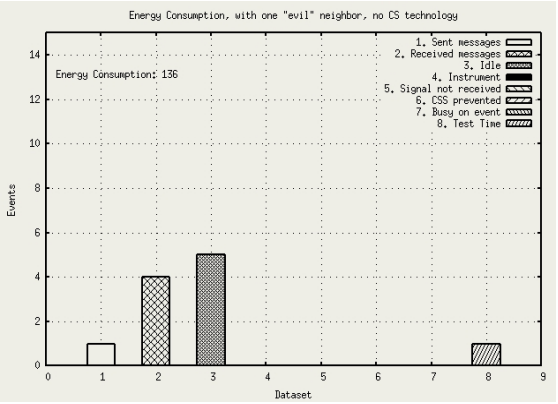
The next test which will be presented is dealing with carrier (CS) sensing technology. The obvious result is that when using carrier sensing technology then evil signals will not arrive at the target node, and less energy will be dissipated. This is shown to be true in the following tests. In the first test in figure 5.2(a) the target node is exposed to a single evil neighbor while it is using CS technology. The result is that the node sends a single message and prevents three signals from using much energy by using CS technology. In the next test in 5.3(b) another evil node is inferred, which results in a higher energy consumptions since CS technology is used more. In test 5.3(c) the CS technology is disabled and the energy consumption from a single evil neighbor is larger than two evil neighbor nodes with CS technology. This is further emphasized in figure 5.3(a). It is thus clear that from an energy saving perspective carrier sensing technology is a very good idea. Finally it is examined what happens if the target node sends very often. The result is shown in figure 5.3(b) and 5.3(c). As expected, the node starts sending all possible messages, since this requires the most energy, and the evil signals arrive less frequent in this worst corner case.



(a) Single Evil Node, target node using CSS technology

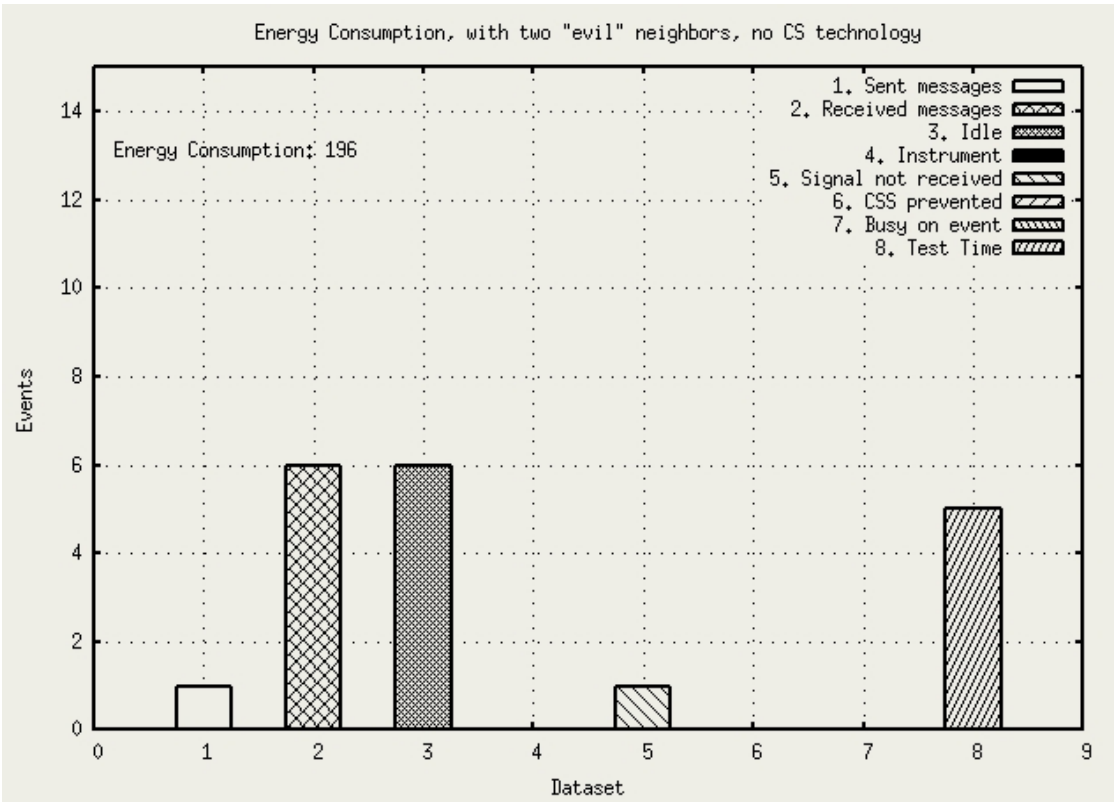


(b) Two Evil Nodes, target node using CSS technology

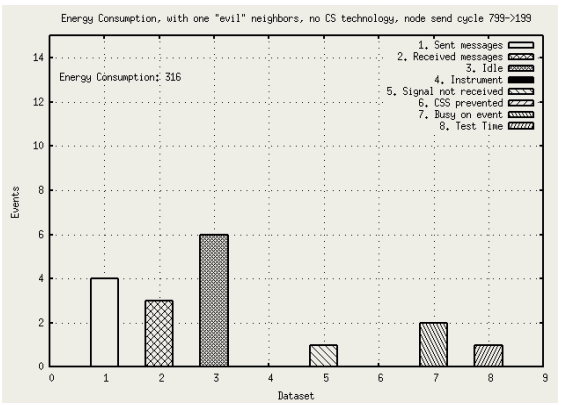


(c) Single Evil Node, target node Not using CSS technology

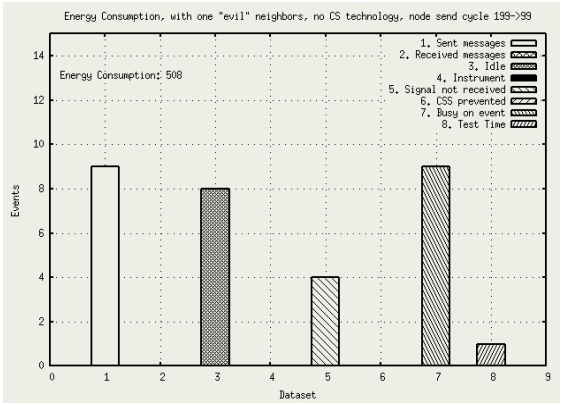
Figure 5.2 *Effect of changing the number of evil neighbors and CS technology 1/2*



(a) Two Evil Nodes, taget node NOT using CS technology



(b) Single Evil Node, with faster target node send cycle, no CS



(c) Single Evil Node, with very fast target node send cycle, no CS

Figure 5.3 *Effect of changing the number of evil neighbors and CS technology 2/2*

5.2.5 Topology

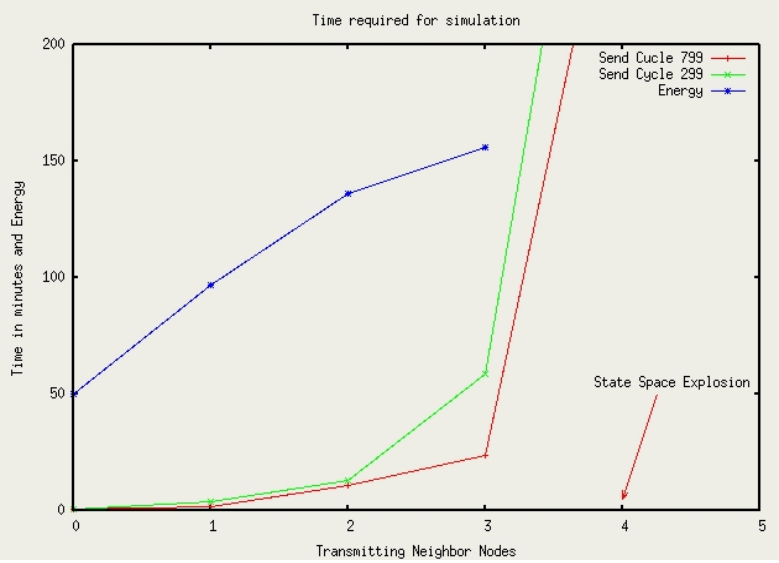


Figure 5.4 *Effect on test time and energy when changing the topology*

The data presented in figure 5.4 is derived from the test results in appendix B.5. In this test the number of neighbors is changed from zero to four and several important lessons are learned. Firstly it is important to notice that the energy does not increase exponentially or linearely as more nodes enter the topology, instead it increases steeply from 1 to 2 nodes and then starts to saturate with three nodes. With five nodes in the topology, (including the target node in the count), a state space explosion occurs and reaches the limits of the available computational power. We can thus conclude that all test must be done with 4 nodes or less in the surrounding node topology.

5.2.6 Varying the Sending Frequency of All Nodes

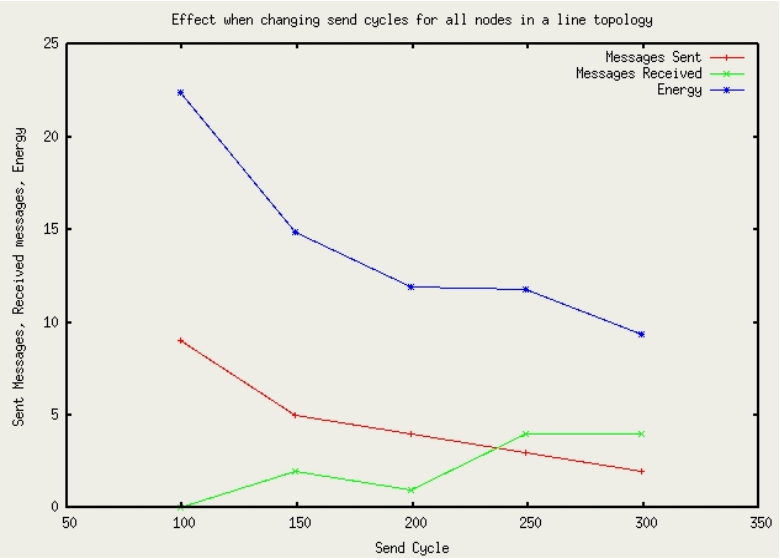


Figure 5.5 *Effect on sent and received messages when changing all send cycles in line Topology*

The data presented in figure 5.5 results from changing the send cycle in all nodes in a line topology. The entire test evolution can be found in appendix B.1. In this test it is worth noting that with the fastest send cycle not a single signal is received, since the node spends it's time sending and thus using the most energy. With a slower send cycle (from 150-250) the energy consumption stabilizes and then begins to fall again. This is a nice example where the formal energy analyzis would tell the designer to examine if the throughput is sufficient with a send cycle around 200 time units. This will allow communication to flow, and avoid high energy consumption.

5.2.7 Instrument usage frequency

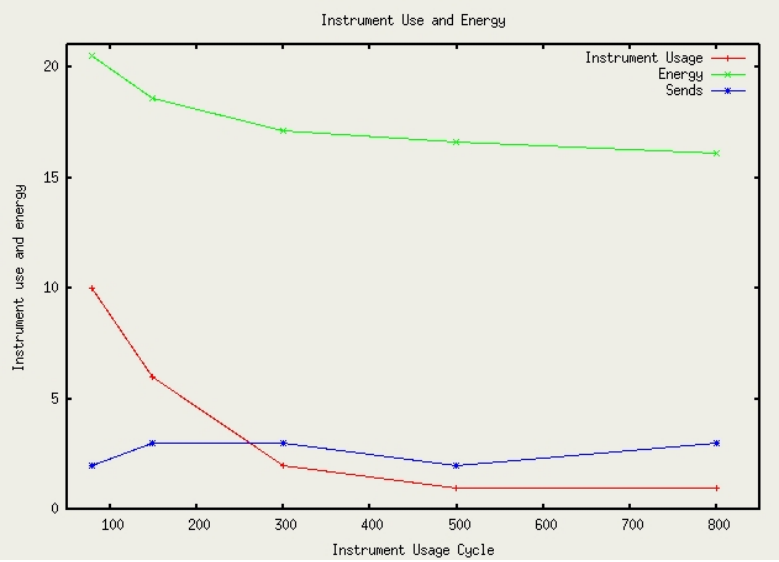


Figure 5.6 *Effect on time and energy consumption when changing the instrument usage cycle*

In this test the effect of increasing the instrument usage frequency is examined. The primary result can be seen in figure 5.6, while the entire dataset is attached in Appendix B.4. The effect on the energy consumption is accumulating with the number of times the instrument is used as expected. This is due to the fact that the energy required to use the instrument is set to 5 while sending and receiving is more expensive. Thus the worst case senario is the basic send/receive scenario with the line topology, only varying within the precision span. The instrument is then simply being used in between. In this test scenario saturation is never reached, so the energy consumption is approximately proportional with the instrument usage.

5.2.8 Message length

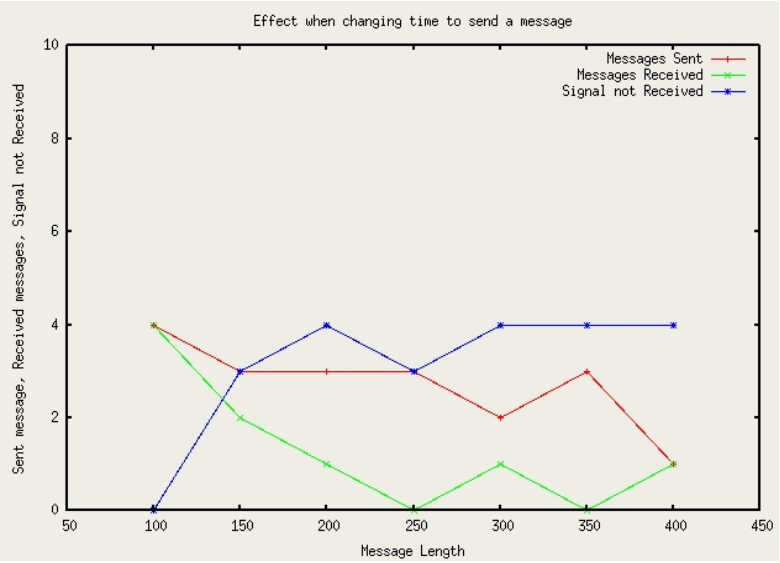


Figure 5.7 *Effect on sent and received messages when changing the message length*

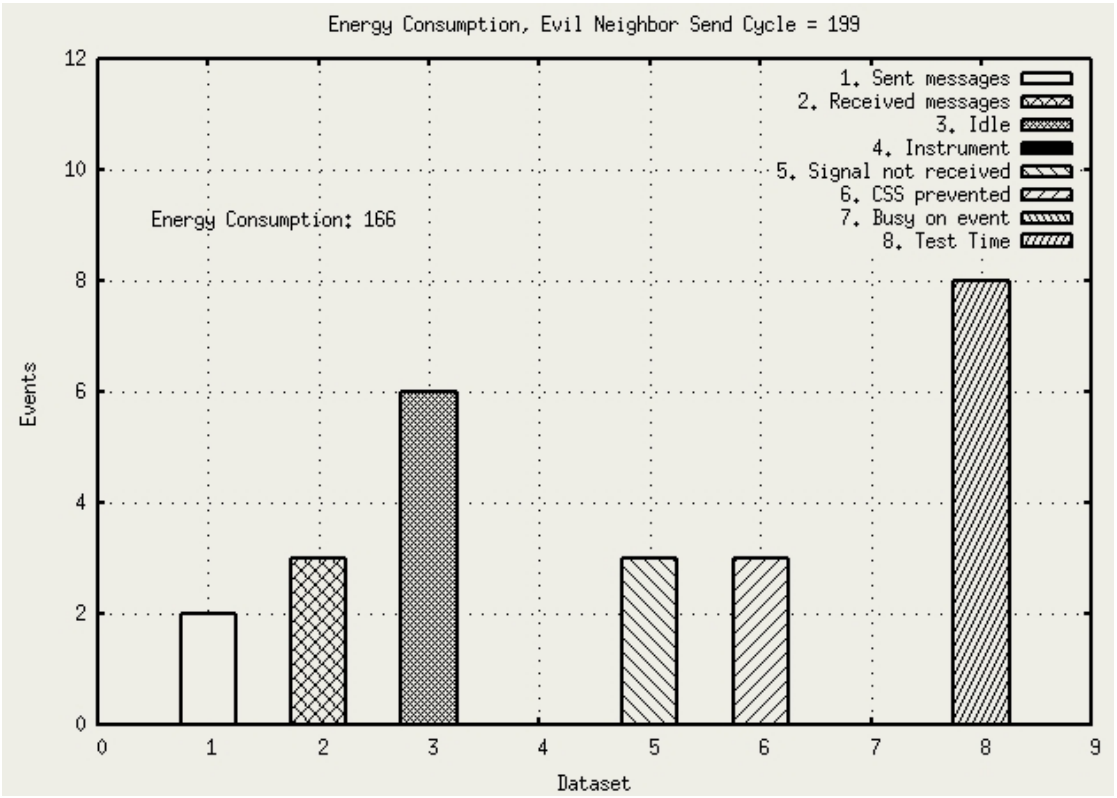
The data in figure 5.7 was taken from appendix B.2. In this test the time to send a message is changed while the energy required for the action remains unchanged. This allow for simulation of high transmission rates against a slow transmission. As expected, a higher time interval required to send and receive a message will cause less messages to be sent and received. It is also noted that the worst case again involves several signals not being received because the surrounding nodes have tried to send messages, which in the worst case was not caught by the node.

5.2.9 Passing traffic

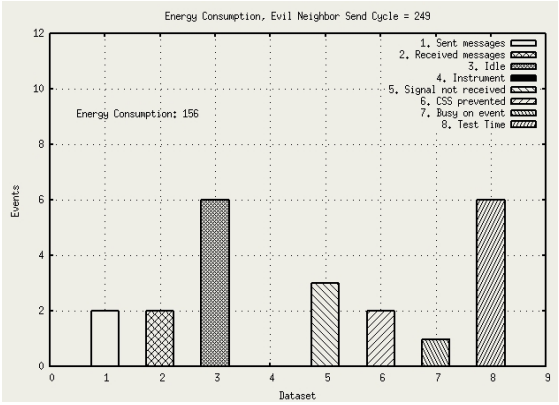
In this final test an evil node will try to spread malice and we examine the effect of varying its send cycle. The evil node is inferred into a normal line topology including a nice node. The result can be observed in figure 5.2.9. As expected the energy consumption falls when the evil signal become less frequent. It could also be noted that the effect is minor, since a 200% increase in evil signals only yields a 6% increase in energy consumption.

Energy Consumption has two components, which must be tested. The first and obvious is the total energy consumption. If no energy harvesting is possible, this is a crucial factor in life-time calculation. The second component is the energy consumption patterns. Finding the spikes in energy consumption allows the system designer to evaluate this against the attached energy source and/or take counter steps to even the distribution of the energy consumption. Using formal model reachability analysis it is possible to find the worst-case behavior in the two cases.

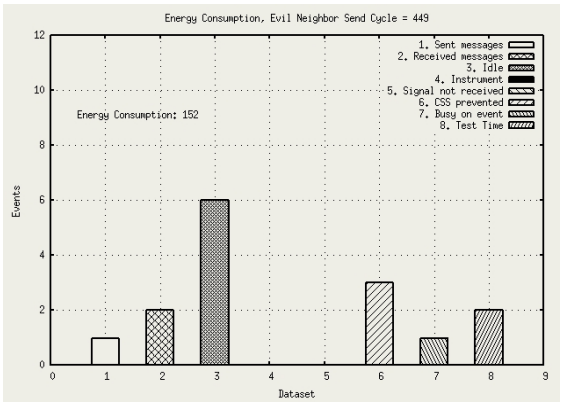
Moving the node closer to the *Base-station* means that the farther away *nodes* will transmit their message through the node. As a consequence, the node will have a higher energy consumption, the closer it is, to the *Base-station*. This is described in figure 5.9.



(a) Evil Node send cycle = 199



(b) Evil Node send cycle = 249



(c) Evil Node send cycle = 499

Figure 5.8 *The effect in a line with an external node with varying send cycle*

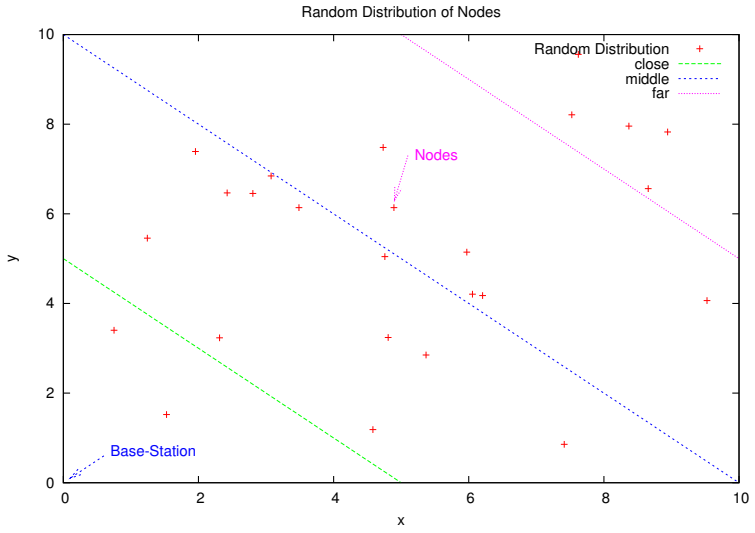


Figure 5.9 *Random distribution of Nodes*

5.2.10 Total Energy Consumption

The Total Energy Consumption in a sensor network using a protocol supporting multi-hop messages and a single implicit modelled base-station is investigated in the analysis.

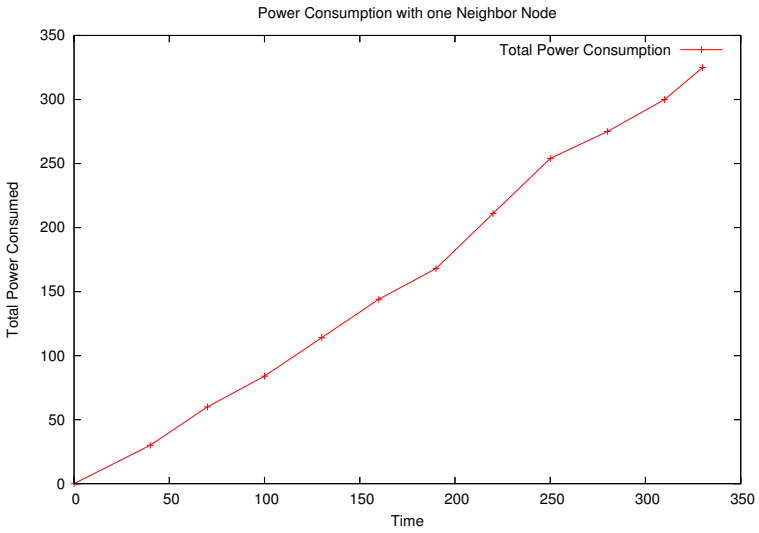


Figure 5.10 *Total Energy Consumption * power should be energy*

In figure 5.10 the total energy consumption of the single node can be observed. By close inspection, the reader will see a change in $\delta energy$ after 200 time units has passed. This is because the node is communicating either with another node or the base-station. From the figure is clear that if the energy source supports 300 energy in 300 time, then the system will live this long.

5.2.11 Average Energy Consumption

The average energy consumption in small time intervals allows for the analysis of spikes in energy consumption. In figure 5.11 the results obtained is presented. In the figure it can be seen that the system never uses more than 23 energy in 40 time.

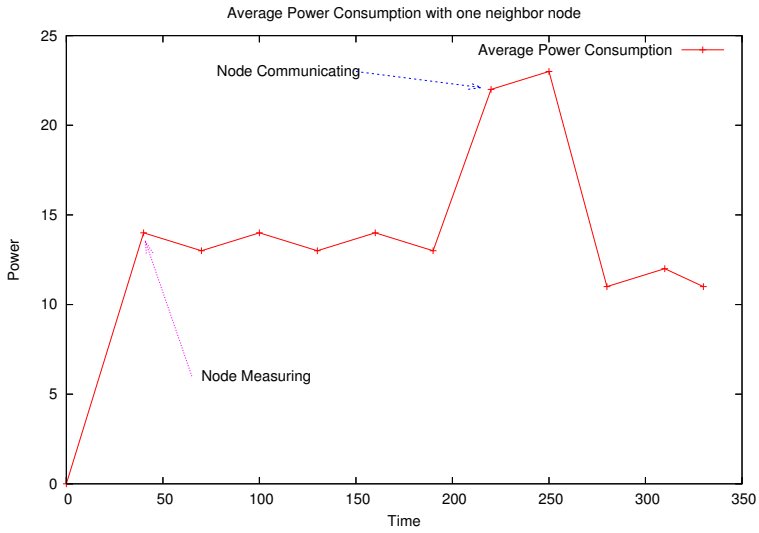


Figure 5.11 *Average Energy Consumption *(the figure should read 'energy')*

5.2.12 Summary

This section started with an introduction to the test strategy and a description of how the tests were performed. Then the actual test were performed on the model. verifying that it worked as expected. First test involved varying the nice neighbor node sending frequency, and then it was tried to vary all nodes sending frequency. The results were presented and acknowledged as correct. Then testing was done on the use of sensing technology and topology changes. Here comments on the time to test and limits in neighbors were discussed. Thirdly the model was told to use its instrument, the message length was changed and the effect of passing traffic examined. Finally energy dissipation over time was analyzed thus including the interesting subject of *energy* patterns in the research.

5.3 Verification

5.3.1 Introduction

Verification of the *END* model involves exposing it to a variety of topology and protocol combinations. This was done in the first part of this chapter. In the next section the result from extending these tests for comparison with what four different authors have presented are discussed: *"S-MAC" (Heidemann),"PAMAS" (Singh et al.), "IEEE 802.11 Multi Hop protocols" (Tseng et al) and "A new protocol" (Neugebauer et al).*

5.3.2 S-MAC Protocol

In this section the effort taken to verify that the SMAC protocol can be correctly modelled is described.

The *S-MAC* introduced in[38] is a choice to be considered for networks with multihop routing. In the introducing article the authors explain and present the results from their experiments with the protocol implemented on UCB MICA motes.
To design the protocol efficiently they identify the following major sources of energy waste: collision, overhearing, control packet overhead and idle listening.
As a measure of how efficient a protocol is in term of idle listening the *idle:receive:send* ratio for IEEE 802.11 is measured to be [1:1.05:1.4] while the specification reads [1:2:2.5]. Spending some time solving this problem should thus be well worth the effort. These kind of ratios are directly translatable to the generic E.N.D model, and the effect of varying these parameters can be examined.
The S-MAC article has a thorough analysis of the latency and throughput involved when using the protocol. It would be obvious to implement and test the analysis results using the *UppAal* framework.
The *S-MAC* protocol infer these ideas in the protocol: Periodic listen and sleep, Collision avoidance, Coordinated sleeping. Using the E.N.D model the following subsubsection will analyze the results from the article

Analyzing the S-MAC protocol with E.N.D

The topology is a two-hop network with two sources and two sinks.

Test 1 - Mean energy consumption on radios in each source node

The main result from this test is that adaptive sleep becomes a better idea the fewer messages are sent. Figure 5.12[1] demonstrates the relationship between the SMAC protocol using adaptive sleep and the 802.11 protocol without sleep. The difference is modelled as an increase in average sleep consumption. As expected we see the same evolution in energy consumption as found in the article. When few packages arrive, the advantage of using S-MAC is very clear while becoming less important as more

[1] based on the test data stored in */tests/smac/messageIntervalComp*

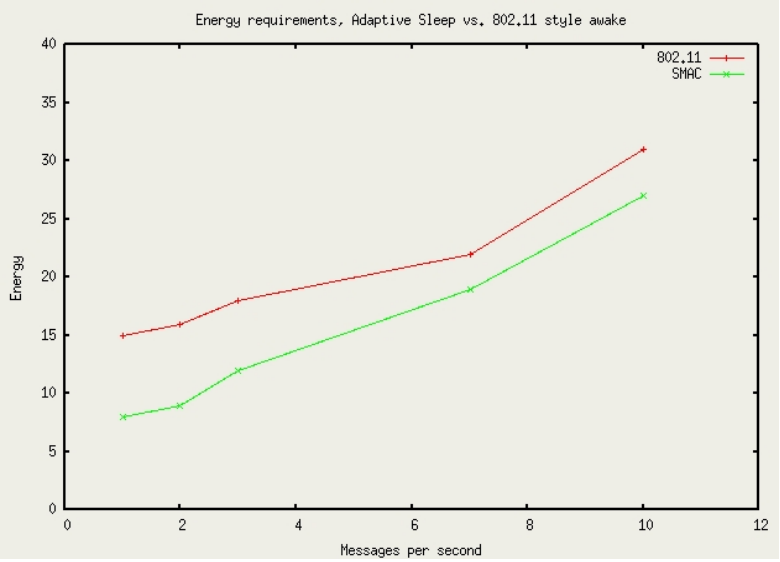


Figure 5.12 *Comparing SMAC to 802.11 using E.N.D*

messages arrive. The reason is that the node is capable of going into deep sleep when messages are not arriving. At the other end of the scale, with 10 messages per test, the node is almost fully occupied and the difference between the protocols become minor.

Summary

Modelling the S-MAC protocol with E.N.D was successful because the same results as shown in the original article was obtained.

5.3.3 PAMAS Protocol

The *PAMAS* protocol combines the idea of a signalling channel with letting the node sleep if it has nothing to do. It is expected that the energy savings are greatest in dense networks since many neighbor nodes will sleep while normally awake. The weakness of the protocol is when high contention for the signalling channel happens, such as in very dense or high traffic - small message networks.

Analyzing the PAMAS protocol

Test 1, Power saved as function of traffic
The first test performed by the *PAMAS* authors deals with about the power saved when the traffic is heavy and light. The result is that the energy savings is generally close to around 50% for a network with 10 or 20 fully connected nodes. The computational power required to simulate the E.N.D model with more than four neighbors is higher than what is currently available at IMM, DTU. Similar results is obtained using only

2-4 neighbors, though. This has already been shown in figure 5.2(b) and figure 5.2(c), where the savings in using carrier sensing technology is $\frac{97}{196} * 100 = 49\%$.

Test 2-4, Power saved as function of Topology - Line, Random, Fully Connected
In a line topology less power is saved, since there are fewer messages which will be overheard. Thus this test is similar to what has already been tested in section 5.2.5 concerning changes in energy consumption with different topology using carrier sensing technology. To perform a test comparable with the result in the *PAMAS* article, a similar test done without carrier sensing technology is needed. This can be seen in figure 5.13. From the figure it is clear that there is no benefit between PAMAS and 802.11 style protocol when the node is alone. When more nodes join the topology the effect becomes clear and is measured between 10% and 35% which complies perfectly with the results in the original article. The data to create the graph is found in */tests/pamas/*.

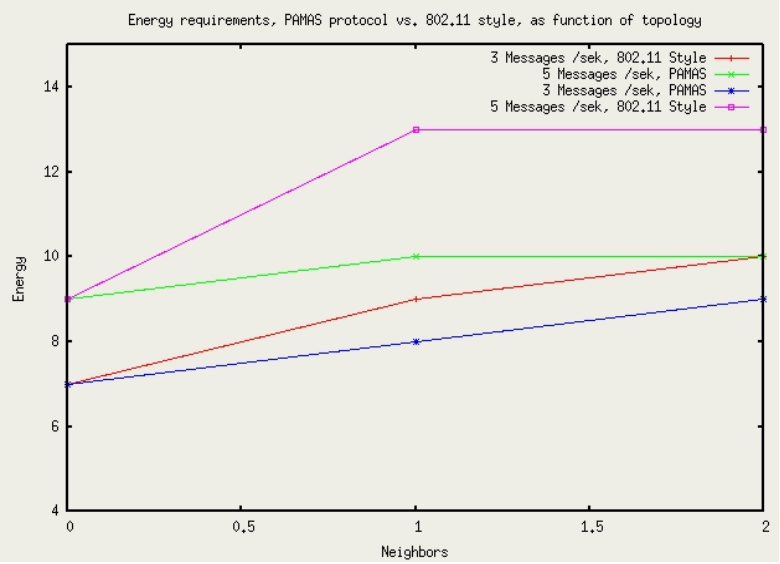


Figure 5.13 *Comparing PAMAS to 802.11 style using E.N.D*

Summary

In this section the *PAMAS* protocol was analyzed with the E.N.D model. The results showed to be similar to what was attained in the original article by Suresh Singh and C.S. Raghavendra.

5.3.4 Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks

In this section it is verified that the *END* model is capable of obtaining the same results as found in article[19] on Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks.

The protocols are based on 802.11[47] but extends the way the Power Saving (*PS*) mode is managed. The following notations are used throughout this chapter: *BI*: length of a beacon interval, *AW*: length of an active window, *BW*: length of a beacon window, *MW*: length of an MTIM window, *SIFS*: Shortest Interframe Spaces .

The protocols work with the beacon interval to sleep as much as possible while still guaranteeing to discover new nodes.

The protocol works like *S-MAC* and *PAMAS* trying to weigh benefits against drawbacks of having the node sleep with it's radio turned off as much as possible. The general power saving results analyzed in section 5.3.2 and 5.3.3 can thus be almost directly transferred. The difference between *PAMAS* and the *802.11 Multihop* lies in the lack of a communication channel - while it is very similar to *S-MAC*. Still the article on *802.11 Multihop* does a very good job of weighing the pros and cons of different sleeping intervals. In this section we will analyze how the worst case energy consumption relates to this analysis.

Dominating-Awake-Interval

This protocol guarantees that a PS host beacon window will always overlap with any neighboring PS host active window in every other interval if $AW \geq BI/2 + BW$ is satisfied. Thus the node will sleep less than half of the beacon interval frames in PS mode.

For the beacon the following parameters is used: *BI*: 100ms, *AW*: 60ms, *BW*: 5ms, *MW*: 5ms.

Periodically-Fully-Awake-Interval

Two types of intervals are possible in PS mode. "Low Power Intervals" (LPI) and "Fully-Awake-Intervals" (FAI). The fully awake intervals appear periodically every T beacon. Thus, this protocol can be modelled as each PS beacon using the average energy:

$$\frac{LPI(T-1)+FAI}{T} \tag{5.1}$$

The protocol guarantees that every T beacon intervals, nodes in reaching distance will become aware of one another. This protocol is better that the dominating-awake when T is larger than 2. Unfortunately using a large value of T makes discovery of new nodes slow. Thus this protocol is best for slow moving environments, where a large value of T is acceptable.

Quorum-Based

Quorum is a minimum set of identities from which one has to obtain permission to perform some action[19]. Quorum defines that the protocol need only be fully-awake every $O(1/n)$ beacon. The advantage is clear when transmission is expensive and nodes enter the network infrequently.

Analysis

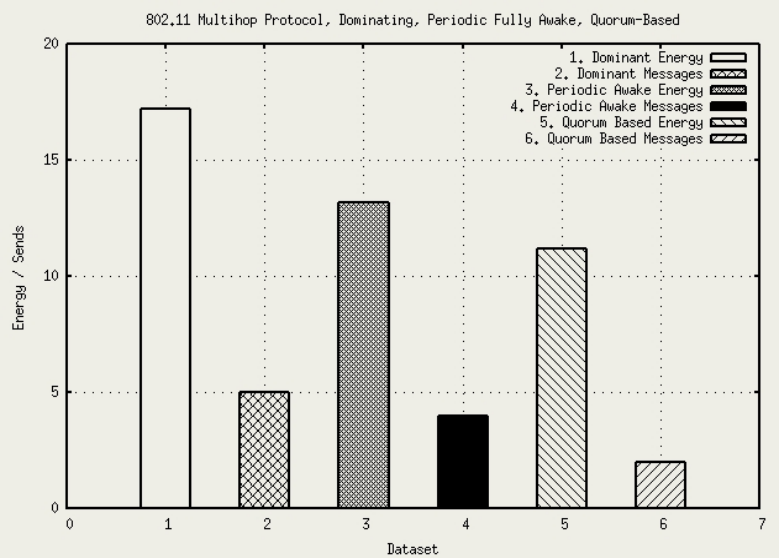


Figure 5.14 *Effect on number of messages and energy consumption*

In this test Power consumption vs. Traffic load is tested. (80% of the traffic load is good traffic). Unicast is used for transmission as opposed to Broadcast. *Message size* is set to 2048 bytes. *Beacon* window size is 8 ms and *MTIM* window size is 16 ms. The active window is MTIM+BW + SIFS. The result of the test is shown in figure 5.14. As expected using the dominant protocol results in the highest energy use, but in return most messages are transmitted.

Summary

In article [47] three different protocols for generating a multi hop network based on 802.11 is proposed. A number of tests are done in the article on power efficiency and power consumption. In this section the general relation between the three protocols proposals are successfully proven correct.

5.3.5 Comparing Protocol Level model with Abstract Model

In this section a comparison of the pros and cons it carried out with the data level *A.N.P* model and the abstract *E.N.D* model.

Comparison

The basic idea of the *A.N.P* protocol is to only send data if data is available. It is not possible to model the details of this idea with the abstract *E.N.D* model. In many cases this won't be necessary though. The same function used to decide whether there is data to

send in the *A.N.P* model, could be used to decide how often data is sent in the abstract model. Using implicit modelling it is therefore easy to translate the data level model to the abstract model. In this case it is a simple matter of decreasing the send cycle. Once the idea has been transferred to the abstract model then the designer has the possibility of defining test scenarios where the worst case corner case is covered fully.
On the other hand the data model yielded the possibility of asking for verification of protocol properties such as *not Deadlock* and *Liveness*. Furthermore, details may be inferred into the model on the designers wish.

Summary

Implementing a protocol correctly as a timed automata is a task requiring a measurable amount of time, especially for designers new to the area. In this case modelling ideas implicitly with the *E.N.D* takes little time and as is proven successful in this chapter. The *E.N.D* model is also capable of analyzing more complex system using a trimmed state space.

5.4 Conclusion

In this chapter the potential of the *E.N.D* model has been throughly mapped. Verifying that the model behaved as expected followed a verification of ideas presented in four papers well known to sensor network researchers.
The limitations of the *E.N.D* model was also discussed. The problems was due to huge state space surpassing the available computational power. This happened when more than 4 neighbors was inferred into the scenario topology.
Finally a comparison between the *A.N.P* data level model and the abstract *E.N.D* model was given. The conclusion was that the data level model could be modelled implicitly using *E.N.D* and the consequences discussed.
The major results are:

- It is possible to model a sensor network with all protocol details and simulated data exchange in an abstract model.
- It is possible to verify proposed protocol properties and behavior.
- Since the model can correctly verify properties is will also predict the consequence of design decisions correctly.

With the *E.N.D* model throughly analyzed and verified the next chapter will be a discussion and conclusion on the work presented in this thesis.

CHAPTER 6

Discussion and Conclusion

In this thesis the possibilities of formal verification of exact protocol behavior and holistic system modelling has been explored. Several results and arguments for their validity have been presented. In this chapter the achieved results will be put into perspective.

6.1 The A.N.P model

The process and results

The *A.N.P* model was the first step in exploring the possibilities in formal verification of sensor networks. The model is true to the proposed protocol and thus implements the necessary data exchange and evaluation procedures. The design process yielded three important discoveries about using the *UppAal* framework and modelling:

- A range of design patterns, not explained by the *UppAal* framework tutorial at that time, were learned.
- Creating a model which is 100% true to a certain protocol and scenario have several disadvantages. Obviously it is not generic and also simulating more than a single node was extremely time consuming since each node would impose the same amount of possibilities and information.
- Manual analysis of variable spans using formal verification is extremely time consuming and requires frequent attention from the tester.

These conclusions formed the basic idea behind the *E.N.D* model. The next model should be generic and fast, and automated analysis should be possible.

Evaluation

The design and verification process of the *A.N.P* model yielded several important results which have been presented to the authors who proposed the *A.N.P* protocol. It verified that the proposed theory was indeed correct and exposed that the need for node synchronization had not been solved or mentioned by the authors.

The *A.N.P* model showed that this way of modelling could be used by designers examining the consequence of tiny changes in a specific protocol. Unfortunately, the state space was extremely big even for small network topologies making formal verification impossible with current computational resources.
The important thing is to notice, that the actual result was to reproduce an already given result and thus verifying the potential in formal analysis outside its regular area. The offspring from the *A.N.P* model is therefore extremely important, and as expected, the grand thesis contributions to sensor network design space exploration are found using the *E.N.D* model.

6.2 The E.N.D model framework

The conclusion drawn from the *A.N.P* model yielded high expectations for the *E.N.D* model framework. A number of challenges were expected to be overcome, and the results from actual usage of the model greatly anticipated.

The E.N.D Model

Designing the *E.N.D* model involved expanding and reusing numerous *UppAal* design patterns. An important achievement was development of a simple but trustworthy OS model using interrupts on the node and changing neighbor nodes with event streams. These model features host the necessary generic nature to support modelling of a variety of systems.
Not all systems can be modelled though. Asynchronous systems are only partly supported and minor protocol details cannot be directly modelled. The node timing behavior must be known *a priori* to modelling and be compatible with the *E.N.D* scheme. The system successfully models a range of ideas but it must be acknowledged, that several ideas will not easily be translated to the *E.N.D* scheme, and compatibility cannot be promised.

E.N.D Usability

The *E.N.D* model successfully analyzed and verified properties from four important papers. Through traces it was also established how the corner cases could happen and system designers can use this information to drastically improve their systems through counter measures.
It may be argued that the corner cases found in the analysis were not all equally likely to happen. A case study will show how the information given by *E.N.D* can still be used:

Send Case

In a trace, the worst case energy consumption happened if *none* of the *many* neighbor signals reached the target node. As a consequence the node focused on dissipating energy through local *Send* actions.

This scenario may be very unlikely - but possible. These surprising situations may very likely have their important part to play in a system, especially in discovering of energy dissipation sinners and with subsequent system re-design. For example the designer of the *Send Case* system would probably have to take a closer look at the energy cost of sending a signal. The designer might try a different protocol, a cluster scheme yielding shorter distances to send or in a different way try to bring the cost of sending a signal. When the new scenario is ready it can again be modelled using *E.N.D* and this cycle continued until a satisfying result is reached. This test pattern is proposed for all future sensor networks, especially when it is critical that the nodes survive a certain amount of time.

The E.N.D framework

The *E.N.D* framework is a superset of the *UppAal* framework, featuring automated test and analysis of *E.N.D* based models. The framework also include the *GUI* which give sensor network designers a structured environment for creating *E.N.D* models and scenarios. The theory behind the framework is explained in chapter 3 while a tutorial can be found in appendix A.

Evaluation

At present the *E.N.D* framework is mature and ready to deliver results. It has already successfully verified several designs and protocols. It has yet to take part in an actual design process such as the *hogthrob* project [24] currently under development. Being used in the *hogthrob* project the *E.N.D* framework would surely play an important factor in system verification, but also the potential in exploring the possibly devastating corner cases would certainly benefit the design process. Playing a part in an actual design and design optimization process would be the natural next step for the model.

6.3 Contributions

6.3.1 Model of Sensor Network

A generic timed automata modelling the energy consumption in sensor network was designed using the *UppAal* framework. The model was tested and verified to work as expected.

6.3.2 Design and Analysis Framework

A framework capable of the following was created:

- Creating sensor network models
- Discovering the maximum consumed energy trace through explore scripts
- Analyzing the trace

6.3.3 UppAal Future Development

Analysis of variable ranges was accepted as a future enhancement to the *UppAal* framework by developer Gerd Behrmann. [1]

6.4 Formal Analysis

6.4.1 UppAal

UppAal, HyTech or Open-Kronos

The *UppAal* framework was the choice for timed automata verifier throughout this thesis. An alternative would have been to use the hybrid technology tool *HyTech* as done in [14]. Both frameworks can be compiled under both *linux and Solaris* which has been vital for this project. Important to the choice was that the *HyTech homepage* has not been updated since February 2003, the *GUI* which the *UppAal* group created for them is from 1996, and the latest article on developing *HyTech* dates back to 1997 [48]. Equally the latest Open-Kronos release is from September 2002 and also have limited ongoing research. In contrast the *UppAal* group continue to develop new releases every 4 months and has flourishing ongoing research with both the industry and academic environment. The choice of *UppAal* as timed automata verifier therefore remains a good choice.

Intelligent State Space Trimming

The challenge with formal modelling is to trim the state space as much as possible. In the following question the *UppAal* framework will potentially have to do a massive amount of unnecessary verification work.

$$E <> Clock < X \&\& Energy > Y \tag{6.1}$$

The problem will occur if the model does not deadlock as soon as the Clock hits X. *UppAal* is not capable of analyzing that the clock will never be reset and that all states where the Clock has passed X are useless. In the *E.N.D* model this is solved by making the model deadlock after the clock has passed the threshold *testTime*. An even better solution may exist. The *ObsSlice* Timed Automata Slicer introduced in [49] has a new perspective on verifying questions which are verified using observers rather than Timed

[1] http://bugsy.dominic.auc.dk/cgi-bin/bugzilla/show_bug.cgi?id=120

Computation Tree Logic *TCTL* formula. According to their research the approach may lead to significant time and space savings during verification. This is because it can discover the set of modeling elements that can be safely ignored at each location of the observer by synthesizing behavioral dependence information, which is exactly what is needed for efficient verification of properties similar to what is done using formula 6.1. This new tool is perfectly capable of interfacing *UppAal* which would still function as the model checker.

6.5 Perspective on other Areas

The *E.N.D* model has demonstrated that a complex heterogeneous systems such as a sensor network can be modelled, verified and analyzed using formal analysis. An area of research with similar properties is Multi Processor System on Chips. Modelling already has a strong influence in the area and event streams have already been proven very efficient by *Kai Richter et. al.* in [13] on *"A Formal Approach to MpSoC Performance Verification"*. The *"Network-Centric System-Level Model for Multiprocessor SOC Simulation"* developed by *Jan Madsen et. al.* in [2] can also be modelled similar to the *E.N.D* model. For the *MpSoC* area to benefit from formal analysis focus will shift from *Energy* to *Throughput*. The potential for *MpSoC* analysis would thus be to answer questions such as: *How good will this network perform in the best case?* and *In the worst case what will happen?* for example in terms of transferred packages. The design of a *MpSoC* model would be very similar to the *E.N.D* model, with event streams implemented as proposed in [13].

6.6 Domains

Formal Analysis has usually been used in the domain of Real-Time systems (*RTS*). In RTS tasks must be performed within strict deadlines. These systems include Embedded controllers, circuits and communication protocols which are all time-dependent systems. The systems will often be a part of a safety-critical application in a complex environment such as aircrafts. These systems have traditionally been extremely difficult to analyze but formal modelling has yielded great advantages in the area. Usually formal analysis is used to determine that the systems will never deadlock and always perform critical actions before their deadline. To analyze these safety-critical questions extremely detailed models are needed. The abstract models and questions regarding the range of variables developed in this thesis, are thus fundamentally different from the usual work on formal analysis.

6.7 Future Work

As Moore's Law continue to prove itself, systems will get smaller, faster, cheaper and richer in number. This will be of great benefit to the sensor network area. The biggest short-term cost savings will be from the integration of RF and micro controllers as is already being attempted by companies like ZigCom and Ember.[2] As radios are also equipped with integrated wireless ports the development of sensor networks will be both cheap and easy. On the other hand the systems will get more complex and analysis equally more important. Moore's law also indicate more computational power for analysis purposes. The computational power available for this thesis yielded a maximum of three simulated neighbor nodes. The state space explosion would then cause the system to run out of resources. The heavy ongoing research in the area of timed automata will also relieve this problem as mentioned in [22].

6.8 Conclusion

In this thesis the pros and cons of using formal analysis for verification of sensor networks have been explored. The *A.N.P* and *E.N.D* model was created which both yielded important results.

It was concluded that the protocol level *A.N.P* model had some advantages for detailed analysis. It was also determined that the state space generated by a detailed network model was prohibitively large and not feasible with the available computational resources. The *A.N.P* model was also determined to be important research since it would have yielded a unified way of modelling between the network simulator *ns-2* and *UppAal*.

The abstract and generic *E.N.D* model places the node as the center of the model instead of the network. The environment such as network and neighbor nodes are modelled using event streams instead of complete nodes and base-stations. The result was a tremendous decrease in test time allowing the model to be analyzed on a strong workstation computer. The *E.N.D* model was tested against a range of formal invariants and scenario cases and finally concluded as a working sensor network model.

The *E.N.D* framework was developed to make modelling, verification and analysis of *E.N.D* based sensor network models faster and automated. The framework is composed of: The *E.N.D* Creator, Explore Scripts and Analyze Script.

Using the *E.N.D* framework the *E.N.D* model was further verified when it was possible to obtain the same results as important articles in the Sensor Network area.

It was concluded that the framework was ready to be used in a real *design, re-design* process, such as the ongoing *Hogthrob* project.

[2] http://www.manufacturing.net/ctl/article/CA490998?spacedesc=industryUpdates

The contribution from this thesis was both to use formal analysis for verification of energy-critical sensor networks, and to create feedback for the *UppAal* group on future improvements.

Bibliography

[1] A. C. et al., "μ-adaptive multi-domain power aware sensors," *http://www-mtl.mit.edu/research/icsystems/uamps/*, 2004.

[2] K. V. Jan Madsen, Shankar Mahadevan, *Network-Centric System-Level Model for Multiprocessor SOC Simulation*. Kluwer Academic Publishers, 2004. Introduces Framwork for Network oc Simulation.

[3] K. P. J. Kahn, R. Katz, "Next century challenges: Mobile networking for 'smart dust'," *International Conference on Mobile Computing and Networking (MOBICOM)*, pp. 271–278, 1999.

[4] J. M. Rabaey, M. J. Ammer, J. L. da Silva, D. Patel, and S. Roundy, "Picoradio supports ad hoc ultra-low power wireless networking," *Computer*, vol. 33, no. 7, pp. 42–48, 2000.

[5] V. R. et al., "Energy aware wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 19, 2002.

[6] M. Neugebauer and K. Kabitzsch, "A new protocol for a low power sensor network," *23 rd, International Performance Computing and Communications, Conference*, 2004.

[7] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, (Atlanta, GA), Sept. 2002.

[8] P. J. et. al, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," *ASPLOS*, 2002.

[9] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, p. 8020, IEEE Computer Society, 2000.

[10] Y. Xu, J. S. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Mobile Computing and Networking*, pp. 70–84, 2001.

[11] S. Singh and C. S. Raghavendra, "Pamas - power aware multi-access protocol with signalling for ad hoc networks," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 3, pp. 5–26, 1998.

[12] V. Bharghavan, A. J. Demers, S. Shenker, and L. Zhang, "Macaw: A media access protocol for wireless LAN's," *Proc. ACM SIGCOMM*, pp. 212–225, 1994.

[13] K. Richter, M. Jersak, and R. Ernst, "A formal approach to mpsoc performance

verification," *Computer*, vol. 36, no. 4, pp. 60–67, 2003.
[14] T. J. K. Sinem Coleri, Mustafa Ergen, "Lifetime analysis of a sensor network with hybrid automata modelling," vol. International Conference on Mobile Computing and Networking, 2002.
[15] B. C. Victor Shnayder, Mark Hempstead, "Simulating the power consumption of large-scale sensor network applications," *Conference On Embedded Networked Sensor Systems*, 2004.
[16] M. Leopold, "Power estimation using the hogthrob prototype platform," Master's thesis, Dept. of Computer Science, University of Copenhagen, 12 2004.
[17] S. S. Wooyoung, "Sens: A sensor, environment and network simulator."
[18] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 126–137, ACM Press, 2003.
[19] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh, "Power-saving protocols for ieee 802.11-based multi-hop ad hoc networks," *Comput. Networks*, vol. 43, no. 3, pp. 317–337, 2003.
[20] M. G. et al., "The network simulator - ns-2," *http://www.isi.edu/nsnam/ns/*, 2005.
[21] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *Computer Communication Review*, vol. 26, pp. 5–21, July 1996.
[22] UppAal, "A tutorial on uppaal," tech. rep.
[23] R. S. et. al, "Lessons from a sensor network expedition," *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, 2004.
[24] J. M. et al., "Hogthrob, networked on-a-chip nodes for sow monitoring," *http://www.imm.dtu.dk/hogthrob/*, 2004.
[25] K. M. Hanen, "Simulation framework for wireless sensor networks," m.sc.ee, Technical University of Denmark, 2004.
[26] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *In Lecture Notes on Concurrency and Petri Nets* (W. Reisig and G. Rozenberg, eds.), Lecture Notes in Computer Science vol 3098, Springer–Verlag, 2004.
[27] P. Krčál, L. Mokrushin, P. Thiagarajan, and W. Yi, "Timed vs time triggered automata," in *Proc. of CONCUR'04.* (P. Gardner and N. Yoshida, eds.), no. 3170 in Lecture Notes in Computer Science, pp. 340–354, Springer–Verlag, 2004.
[28] M. J. G. Jan Madsen, Kashif Virk, *A SystemC-Based Abstract Real-Time Operating System Model for Multiprocessor Systems-on-Chips*. Morgan-Kaufmann Publishers, 2004.
[29] SHIFT, "The hybrid system simulation programming language," *http://www.path.berkeley.edu/shift*.
[30] H. Sun, "Timing constraints validation using uppaal, schedulability analysis," *DIPES*, 2000.
[31] A. M. K. Cheng, *Real-Time Systems, Scheduling, Analysis, and Verification*. Wiley-Interscience, 2002.
[32] G. B. et. al, "A tool architecture for the next generation of uppaal," *http://www.it.uu.se/research/reports/2003-011/*.

[33] A. Jantsch, *Modeling Embedded Systems and SoCs, Concurrency and Time in Models of Computation*. Morgan Kaufmann Publishers, 2004.

[34] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Compact data structures and state-space reduction for model-checking real-time systems," *Real-Time Syst.*, vol. 25, no. 2-3, pp. 255–275, 2003.

[35] V. A. Braberman, D. Garbervetsky, and A. Olivero, "Improving the verification of timed systems using influence information," in *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 21–36, Springer-Verlag, 2002.

[36] Abramson, "The aloha system–another alternative for computer communication," *Proceedings of the AFIPS Fall Joint Computer Conference*, vol. 37, pp. 281–285, 1970.

[37] D. J. Khan, "Aloha protocol," *http://murray.newcastle.edu.au/users/staff/jkhan/ALOHA.pdf*.

[38] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated, adaptive sleeping for wireless sensor networks," *IEEE Transactions on Networking*, vol. 12, 2004.

[39] C. Wu and V. Li, "Receiver-initiated busy-tone multiple access in packet radio networks," in *Proceedings of the ACM workshop on Frontiers in computer communications technology*, pp. 336–342, ACM Press, 1988.

[40] B. Karp and H. T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *Mobile Computing and Networking*, pp. 243–254, 2000.

[41] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," in *Mobile Computing and Networking*, pp. 85–96, 2001.

[42] J. S. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," in *Symposium on Operating Systems Principles*, pp. 146–159, 2001.

[43] X. Li, Y. J. Kim, R. Govindan, and W. Hong, "Multi-dimensional range queries in sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 63–75, ACM Press, 2003.

[44] A. Woo, S. Madden, and R. Govindan, "Networking support for query processing in sensor networks," *Commun. ACM*, vol. 47, no. 6, pp. 47–52, 2004.

[45] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Commun. ACM*, vol. 47, no. 6, pp. 53–57, 2004.

[46] M. Rahimi, H. Shah, G. Sukhatme, J. Heidemann, and D. Estrin, "Studying the feasibility of energy harvesting in a mobile sensor network," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Taipai, Taiwan), pp. 19–24, IEEE, May 2003.

[47] "Ieee 802.11, the working group setting the stands for wireless lans," *http://grouper.ieee.org/groups/802/11/*, 2004.

[48] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HYTECH: A model checker for hybrid systems," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1–2, pp. 110–122, 1997.

[49] V. B. et. al., "Obsslice: A timed automata slicer based on observers,"

http://dependex.dc.uba.ar/obsslice/, 2004.

CHAPTER 7

Appendix

APPENDIX A

E.N.D Tutorial

A.1 Introduction

In this tutorial a fast paced introduction is given to the *UppAal* and *E.N.D* framework. An example model is created and it is shown how test and analysis is performed.

A.2 Tutorial

A.2.1 UppAal GUI

The *UppAal* framework can be downloaded from *http://www.uppaal.com*. The framework is normally interfaced using a graphical user interface. In this chapter a short introduction to this environment is presented which should give a basis for understanding the work in this thesis and how to use the *E.N.D* model. A complete tutorial on using the *UppAal* framework can be found in [22]. The *UppAal* GUI can be divided into three parts which will now be presented: design, simulation and verification.

Design

The design environment used to create *UppAal* models is demonstrated in figure A.1. At the top of the screen several menus yield access to model load/saved/print etc. Also a range of parameters can be configured through these menus. Below the menu a toolbar offers the tools necessary for modelling: *Add Location* and *Add Transition*. Below the toolbar three panes offer the possibility to switch between the *System Editor*, Simulator and Verifier. In figure A.1 the *System Editor* is chosen. To the left the created models which will later be concurrently simulated can be switched between. In the figure the *nicenb* is chosen, and to the right the model is displayed and can be edited.

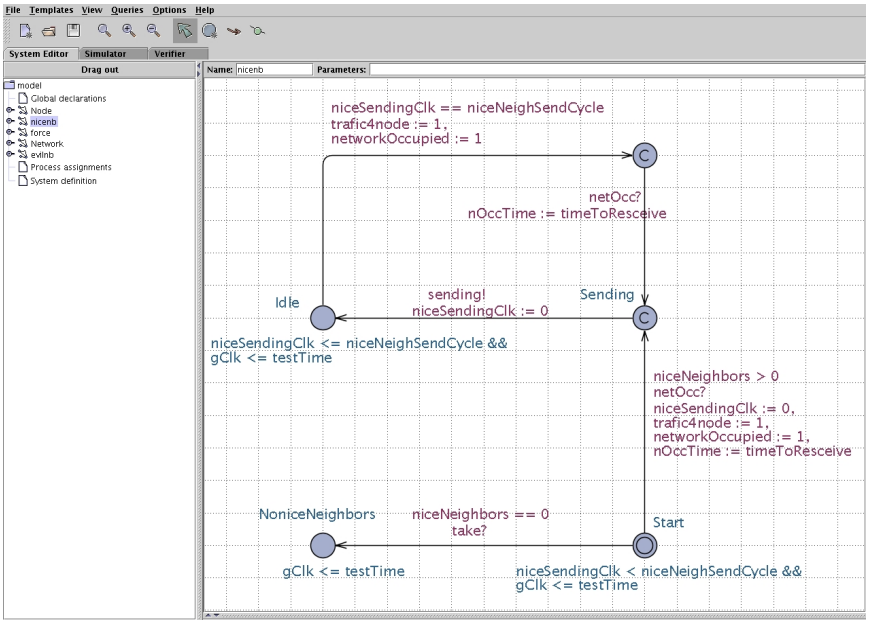


Figure A.1 *UppAal Model Creation and Design*

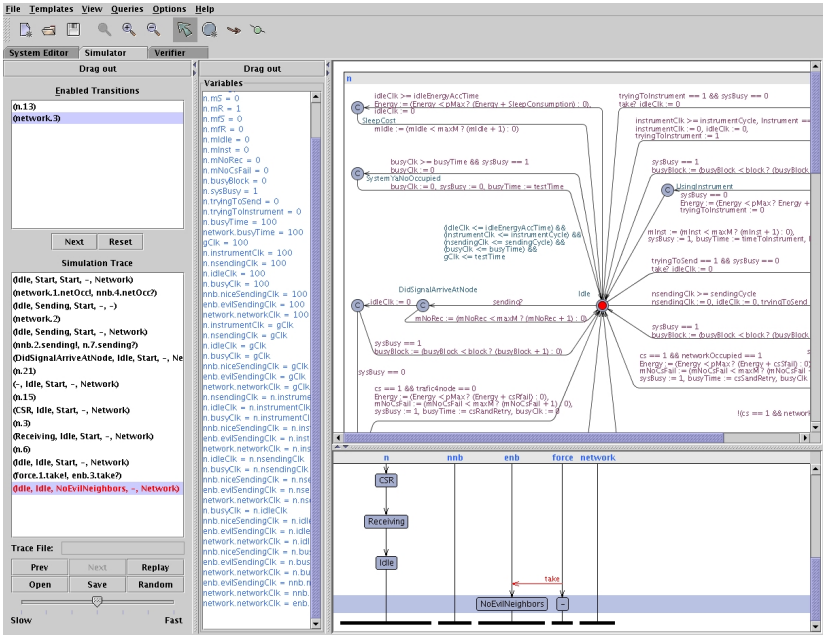


Figure A.2 *UppAal Simulation Environment*

Simulation

The *UppAal* simulation environment has the same menus and toolbar as the editor environment. The content of the pane has shifted to facilitate simulation of the system. In

figure A.2 the pane is shown. At the top left of the pane the available transitions can be chosen and below a trace is displayed. In the center of the figure the current value of clocks and variables are shown. To the right a UML a sequence diagram gives a visual impression of the trace. On top of this the current model transition is displayed.

Verification

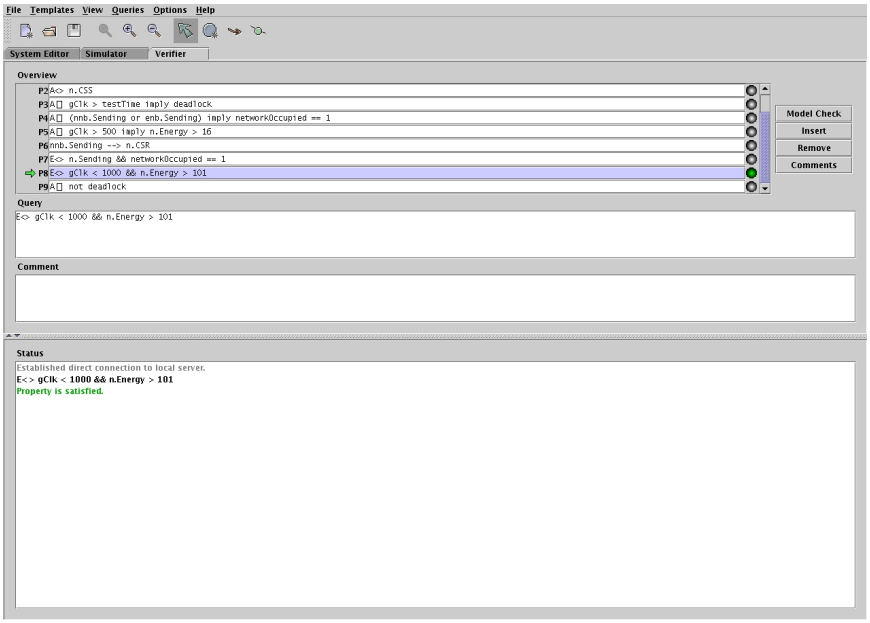


Figure A.3 *UppAal Verification Environment*

The Verification pane allows for formal verification of the created system. In figure A.3 a number of query examples can be seen, and they are tested by hitting the *Model Check* button. If the question yields a trace, this can be followed by returning to the simulation pane.

A.2.2 The E.N.D Modeler

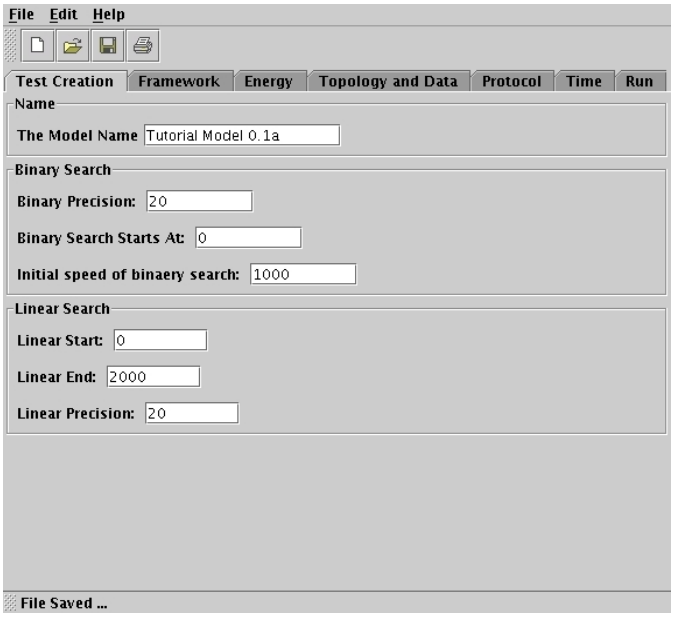


Figure A.4 *EndModel Test Creation*

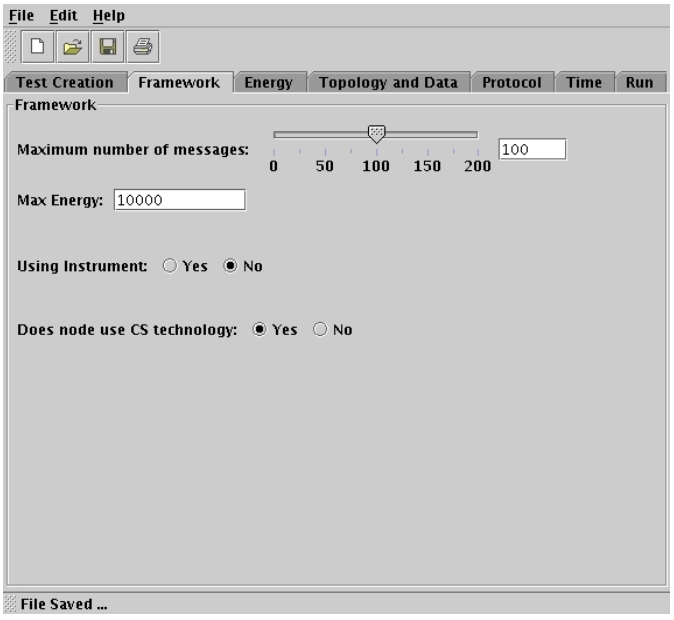


Figure A.5 *Framework*

Test Creation

The first pane in the *E.N.D* modeler allows the designer to customize the binary or linear search script which will later be used. This pane is shown in figure A.4.

Framework

The variables in the *Framework* pane in figure A.5 is defined to do the following. The maximum number of messages should be larger than highest amount of messages/-transmission that will happen through the test time. Equally the *Max Energy* variable should be the higher than the maximum accumulated energy in the test time. Binding these integer model values will allow the *UppAal* framework to make verification faster. The last two values *Using Instrument* and *use CS technology* is specifying weather the node make use of these features.

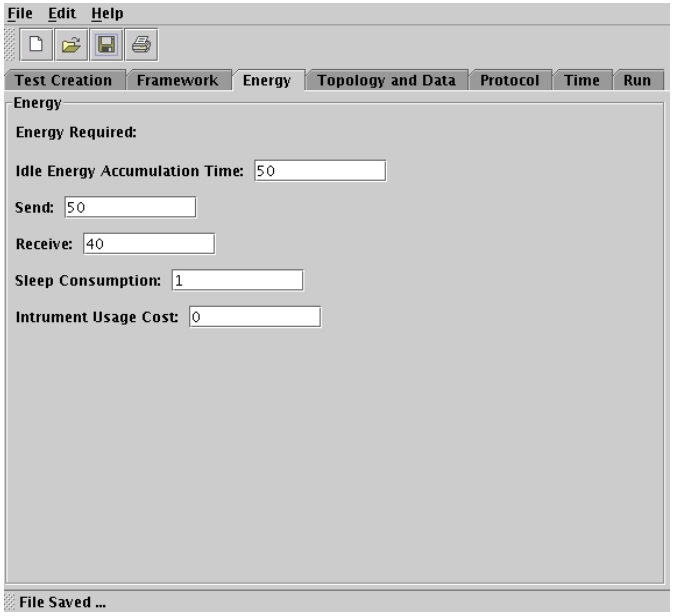


Figure A.6 *Energy*

Energy

Figure A.6 shows the variables which should be set to the amount of energy each of the actions: *Send, Receive, Sleep, Instrument*. The amount of time required before sleep consumption is used is set through the *Idle Time* variable.

Topology and Data

The pane where the topology and the *UppAal* path is specified is shown in figure A.7. The nice neighbors are neighbors which are trying to send data to the node, while an

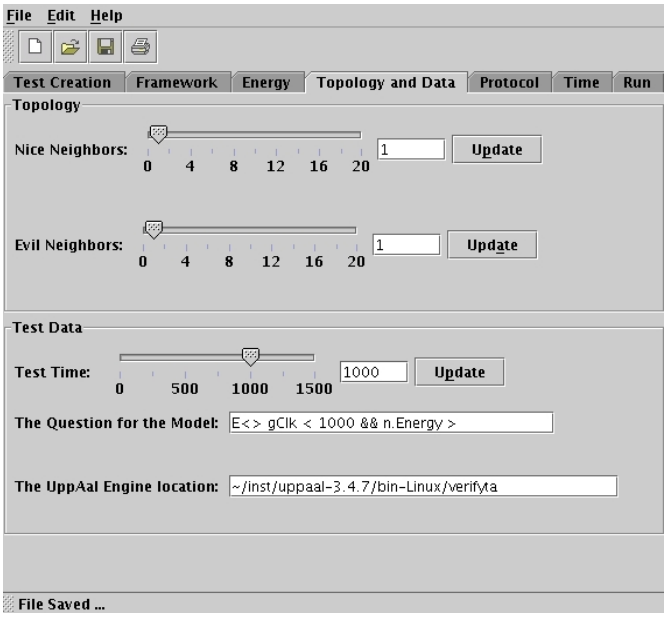


Figure A.7 *Topology and Data*

evil neighbor is simply generating network traffic. The *Test Time* variable defines the length of the test. The *Model Question* will rarely have to be changed, but it is necessary to define the where the *UppAal* application *Verifyta* is situated on the system.

Protocol

The protocol pane shown in figure A.9 is where a system designer does the actual translation from a specific protocol to the *E.N.D* energy accumulation protocol defined in figure A.9 and explained in detail in chapter 3.

Time

The *Time* variables are set in the pane displayed in figure A.10. They define how much time the system will be occupied upon performing a certain action.

Run

The final pane named *Run* is where the model and test scripts are created. This pane is shown in figure A.11. The first button: *Create Model* will create an *UppAal* compatible model defined through the specified variables. The button: *Generate Binary Test Script* will ask for a name and then create a test script for performing a binary test to find the maximum consumed energy in the test time interval. The final button: *Generate Linear Test Creation Script* will generate the linear test creation script which is used to create

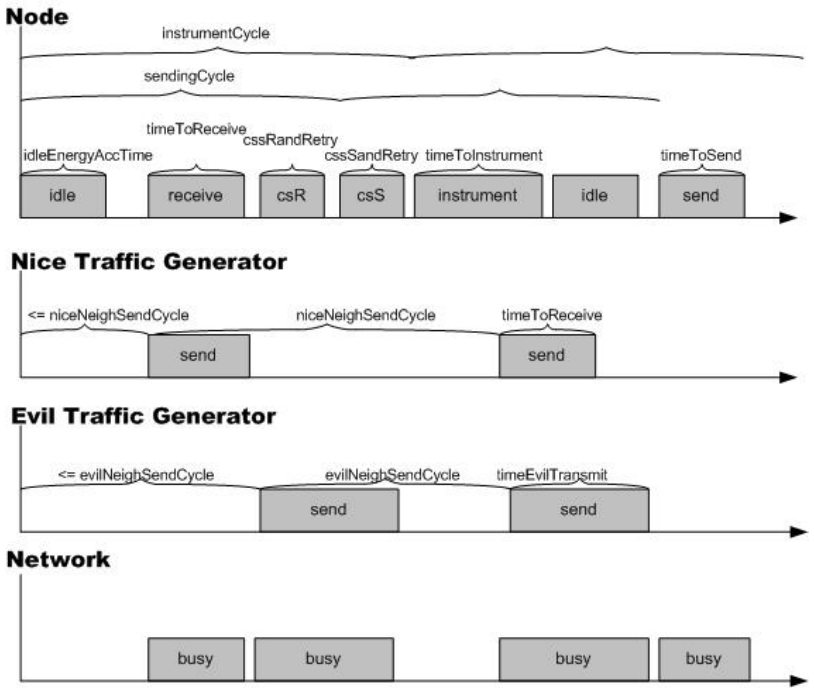


Figure A.8 *An example run of the model defining the necessary parameters*

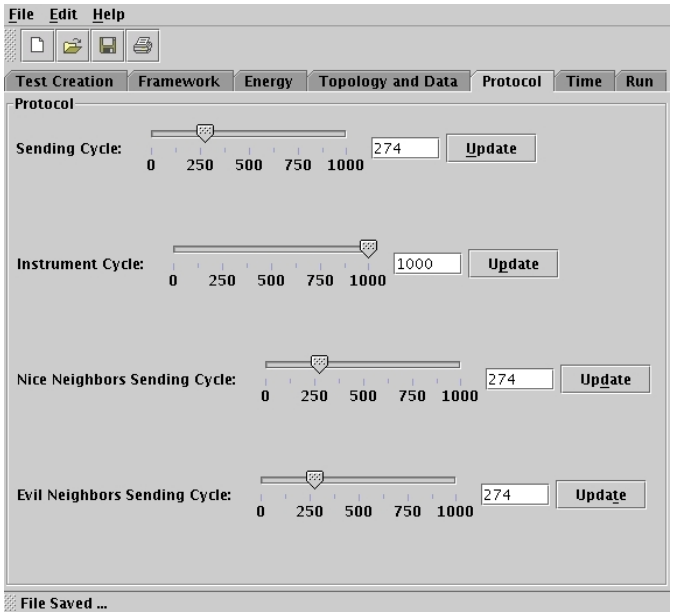


Figure A.9 *Protocol*

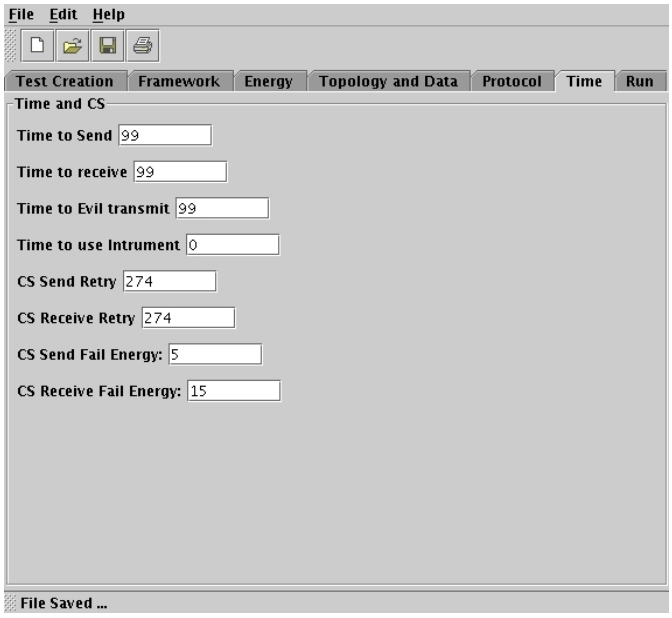


Figure A.10 *Time*

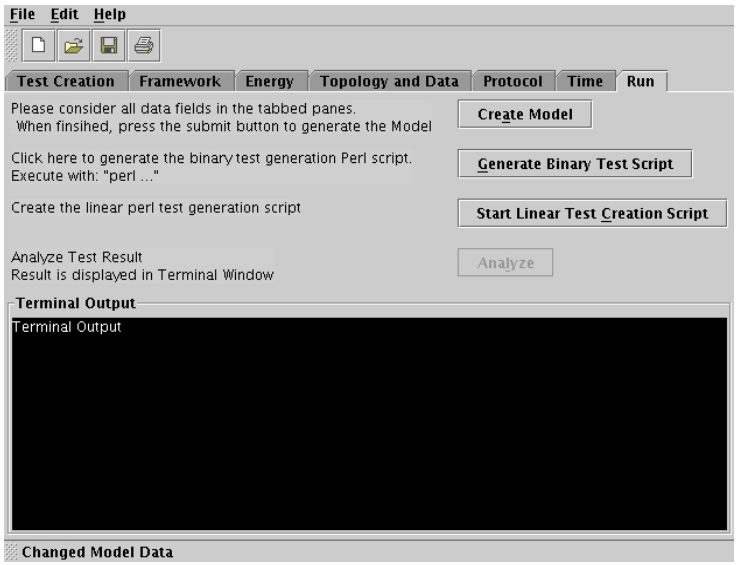


Figure A.11 *Run*

the file *magicQuery.q* which can be used to start a linear search. This will be shown in the subsection A.2.4.

A.2.3 Binary Search

In this subsection it is shown how to find the maximum energy the model created with the data specified in subsection A.2.2 can consume using the binary search script. The model was saved in the file *newmode.xml* and the binary search script to the file *binarySearch.pl*:

A binary search

```
1
2  user@server (~/endmodel/tests/) > perl binarySearch.pl newmode.xml
3  ************************
4  Starting Search...
5  ************************
6
7  2005-02-23 22:30:49
8
9  touch tmp/result \&\& rm -f tmp/result \&\&
10       ~/inst/uppaal-3.4.7/bin-Linux/verifyta -T -f trace -t0 newmode.xml
11       query.q > tmp/resultUPPAAL version 3.4.7, Oct 2004 -- verifyta.
12
13 Copyright (c) 1995 - 2004, Uppsala University and Aalborg University.
14 All rights reserved.
15 Writing example trace to trace-1.xtr
16
17 2005-02-23 22:30:50
18
19 Query has run ...
20
21 ! Uses More Energy, Starting binary search pattern !
22
23 -1-> Updating MaxEnergy from 0 to 100 <-1--
24
25 2005-02-23 22:30:50
26
27 touch tmp/result \&\& rm -f tmp/result \&\&
28       ~/inst/uppaal-3.4.7/bin-Linux/verifyta -T -f trace -t0 newmode.xml
29       query.q > tmp/resultUPPAAL version 3.4.7, Oct 2004 -- verifyta.
30 Copyright (c) 1995 - 2004, Uppsala University and Aalborg University.
31 All rights reserved.
32 Writing example trace to trace-1.xtr88 states
33 2005-02-23 22:30:50
34
35 Query has run ...
36
```

```
Uses More than 100

-1-> Updating MaxEnergy from 100 to 200 <-1--

[CUT]
....
[CUT]

*************************************************************
The system uses more than 322 and less than 323
*************************************************************
```

The result from this search is a trace file which can be used in two ways. The trace can be loaded in *UppAal* and followed as a sequence diagram. The obtained data can also be extracted and saved using the *AnalyzeFile.pl* script as is shown in subsection A.2.5.

A.2.4 Linear Search

In this subsection it is shown how the linear search pattern can also be used to find maximum consumable energy in the *test time* interval. The first part is to create the query file with all the questions in the linear search. The linear script was saves to the file *createLinear.pl* in subsection A.2.2.

Creating Query for linear Search

```
user@machine (~/endmodel/tests) > perl createLinear.pl
10\% done
20\% done
30\% done
40\% done
50\% done
60\% done
70\% done
80\% done
90\% done
100\% done
Test Took: 00:00:03 (hr:min:sek)
 Use the created query like this: uppaal-3.4.7/bin-SunOS/verifyta
             -d-S0 -T -f traces/traceT -t0 model.xml magicQuery.q
```

When the file containing the queries *magicQuery.q* is ready, the test can be started. This is done as follows:

A linear search

```

user@mache (~/endmodel/tests/) >
~/inst/uppaal-3.4.7/bin-Linux/verifyta -d -S0 -T -f traces/traceT -t0
```

```
                                                    newmode.xml magicQuery.q
UPPAAL version 3.4.7, Oct 2004 -- verifyta.
Copyright (c) 1995 - 2004, Uppsala University and Aalborg University.
All rights reserved.
Options for the verification:
  Diagnostic trace is on
  Search order is depth first
  Using no space optimisation
  Using reuse optimisation
  Using cheap inclusion checker.
  Passed list size is 65536
  State space representation uses minimal constraint systems
Property 1 (line 1) is satisfied.
Writing example trace to traces/traceT-1.xtr
Property 2 (line 2) is satisfied.
Writing example trace to traces/traceT-2.xtr
Property 3 (line 3) is satisfied.
Writing example trace to traces/traceT-3.xtr
Property 4 (line 4) is satisfied.
Writing example trace to traces/traceT-4.xtr
Property 5 (line 5) is satisfied.
Writing example trace to traces/traceT-5.xtr
[CUT]
...
[CUT]
Property 15 (line 15) is satisfied.
Writing example trace to traces/traceT-15.xtr
Property 16 (line 16) is satisfied.
Writing example trace to traces/traceT-16.xtr
Property 17 (line 17) is satisfied.
Writing example trace to traces/traceT-17.xtr
Property 18 (line 18) is NOT satisfied.
Property 19 (line 19) is NOT satisfied.
Property 20 (line 20) is NOT satisfied.
...
[CUT]
```

In this example the final trace will be called *traces/traceT-17.xtr*. Which can be used for analysis in the same way explained in the final part of subsection A.2.3.

A.2.5 Analysis

A detailed analysis of a trace file can be examined using the *UppAal* sequence diagram. But is it desired to find the tendency when comparing different test scenarios only the

number each specific action has been perform has to be stored. This can be done automatically by calling the *analyzeFile.pl* script on a trace file. An example of this is now shown:

Analysis

```
user@machine (~/endmodel/tests/) > perl analyzeFile.pl model.xml
                  traceT-17.xtr aResultFileName "Trace Caption" 11
```

This will append the following data to the file *result/aResultFileName*. The content will then look as follows:

result/tracefileName content

```
-------------------------------------
Result: Trace Caption
-------------------------------------
n.Energy = 216
n.mS = 2
n.mR = 3
n.mIdle = 6
n.mInst = 3
n.mNoRec = 0
n.mNoCsFail = 0
n.busyBlock = 1
TestTime = 11
```

It can finally be concluded that the designed model, in the specified scenario, in the specified test time interval, will use a maximum of 216 energy units. The case where this happens involves the following actions: The node has sent two messages and received three. The system has been idle 6 times and it has used the instrument three times. It has never happened that a neighbor signal did not arrive at the node, and carrier sensing technology never caused a fail. When one event happened the system was busy. The system took 11 minutes to be verified.

A possible conclusion on this data could be that carrier sensing technology should be inferred so that *evil* signals would not be fully received causing a waste of energy. This issue could be resolved and the test run again and the results compared. This concludes the design / -redesign cycle facilitated by the *E.N.D* framework.

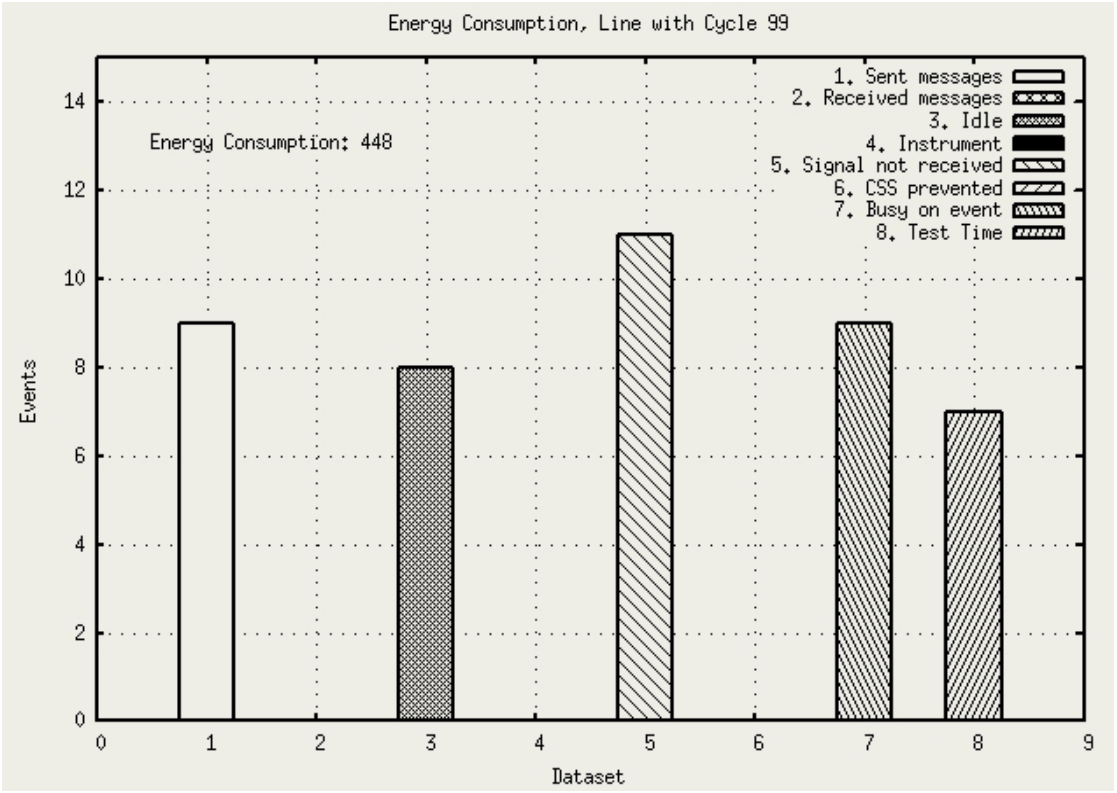
A.3 Conclusion

In this tutorial it has been shown how to create a sensor network model using the *E.N.D* modeler. It was then explained how test and analysis can be performed. Finally an example of data interpretation and the design cycle of changing system parameters and comparing the result was explained.

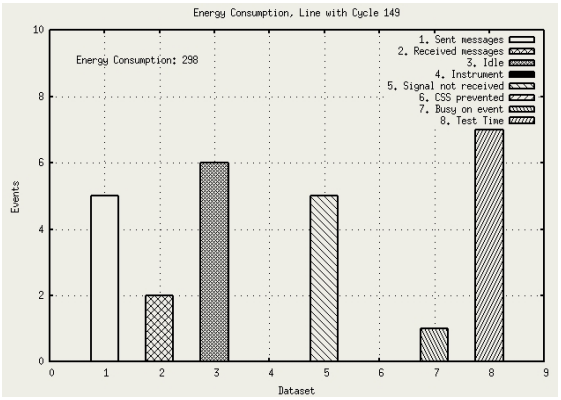
APPENDIX B

Test data

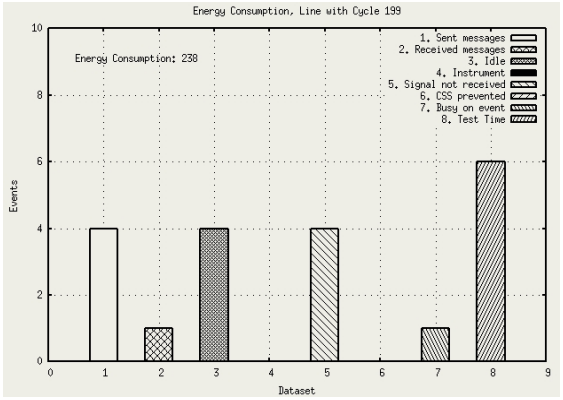
B.1 Changing all cycles in all nodes



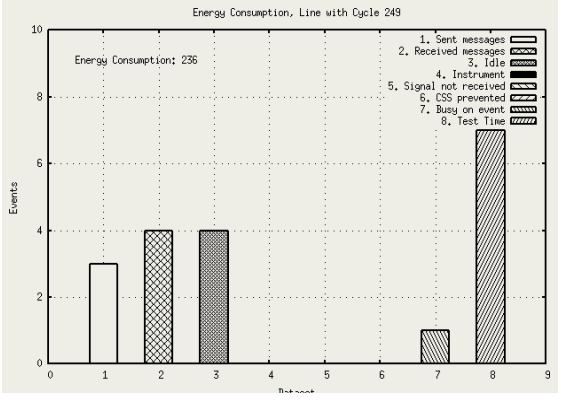
(a) Send cycle = 99



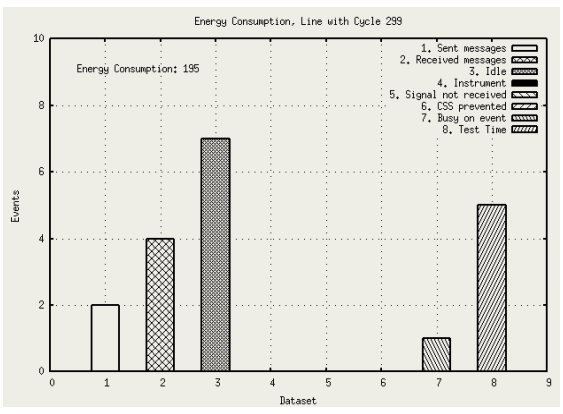
(b) Send cycle = 149



(c) Send cycle = 199

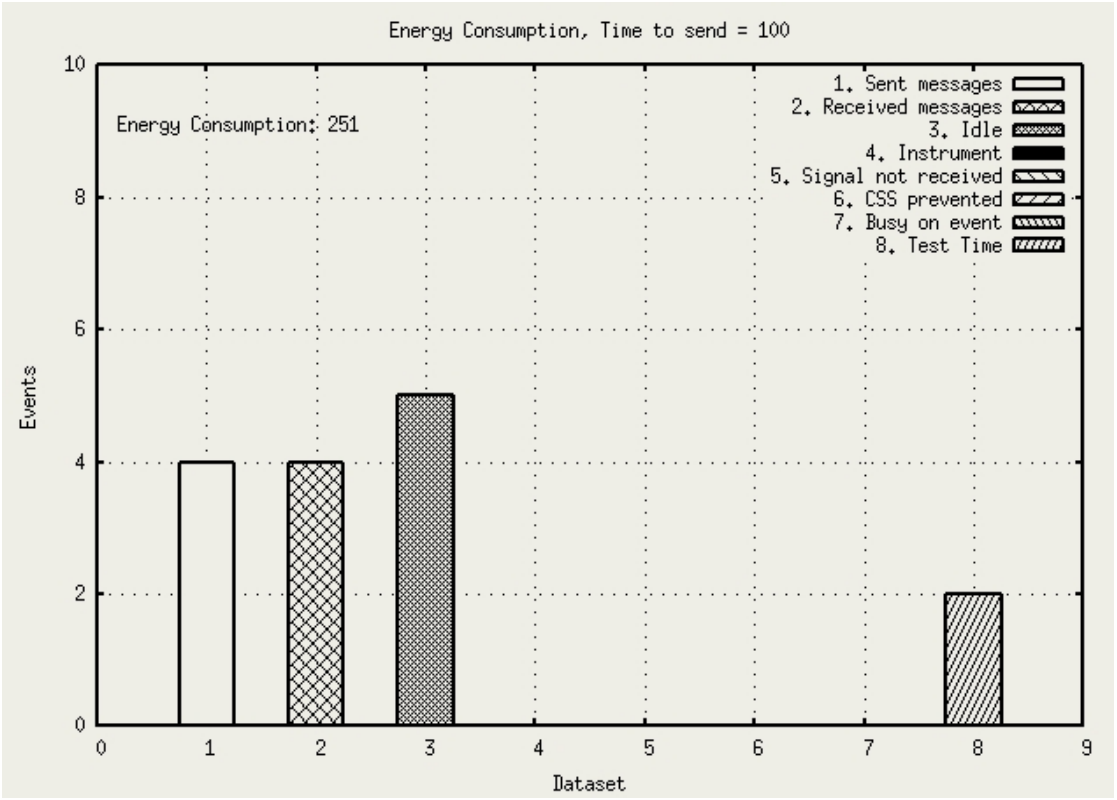


(d) Send cycle = 249

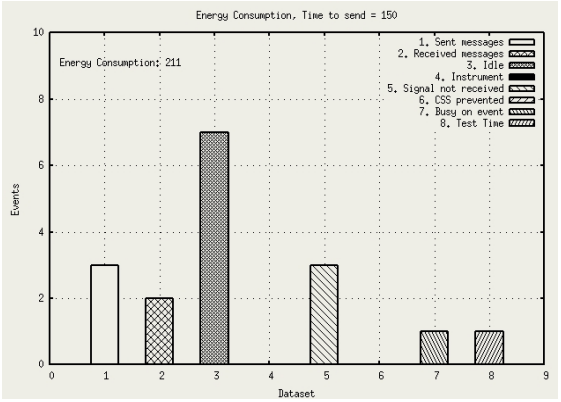


(e) Send cycle = 299

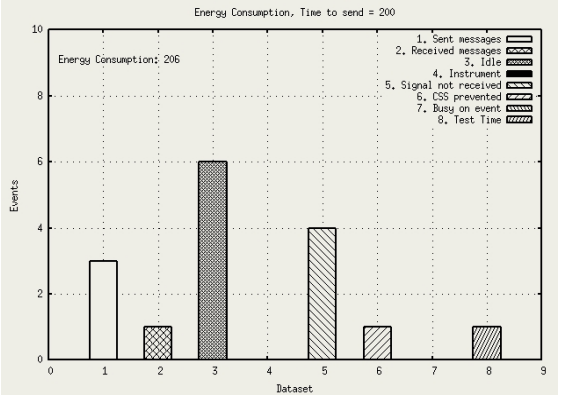
B.2 Changing message lenght in line topology



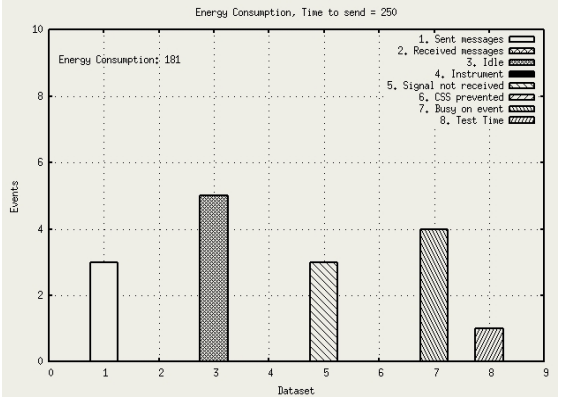
(a) Message length = 100



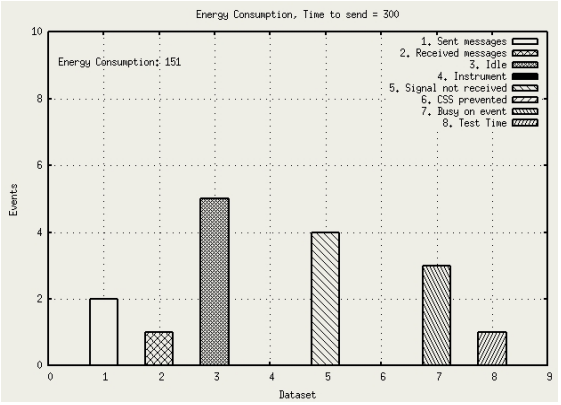
(b) Message length = 150



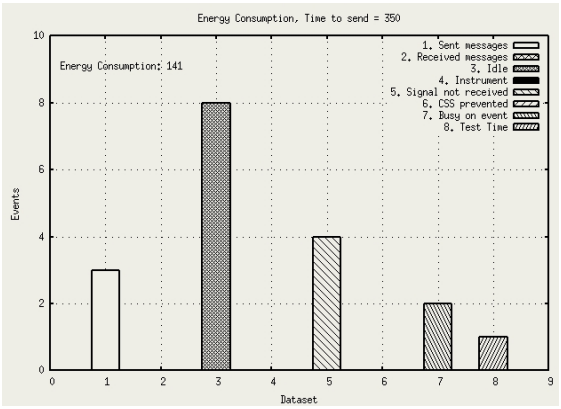
(c) Message length = 200



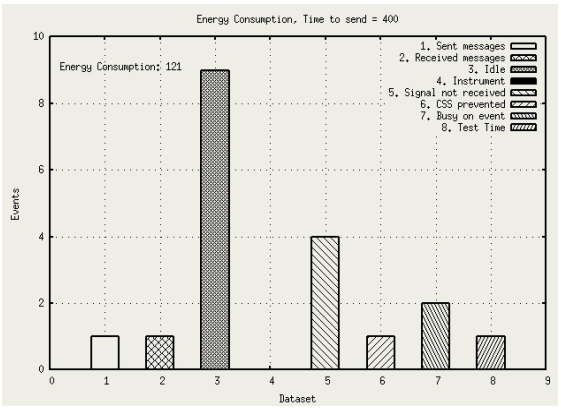
(d) Message length = 250



(e) Message length = 300



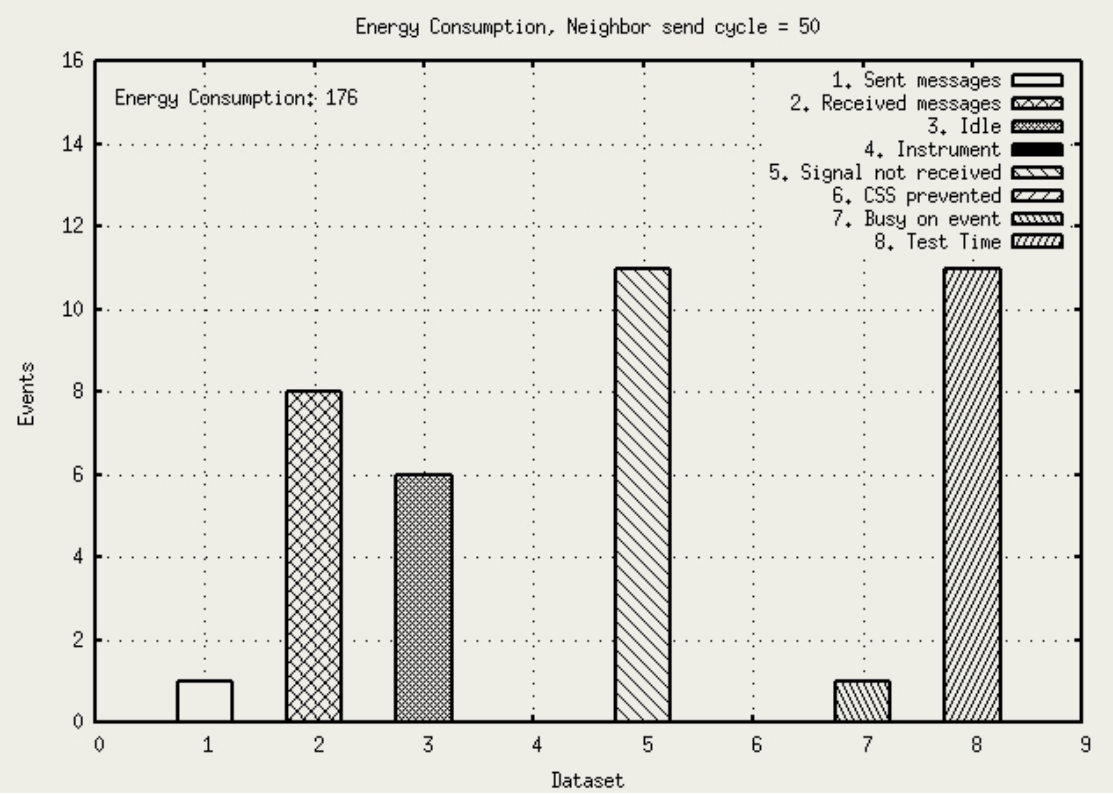
(f) Message length = 350



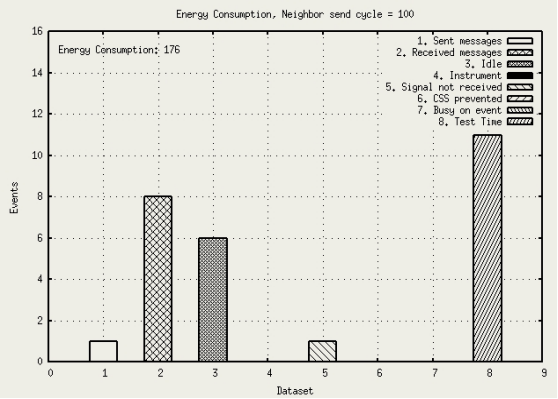
(g) Message length = 400

Figure B.2 *Effect of changing the message length in line topology of all nodes*

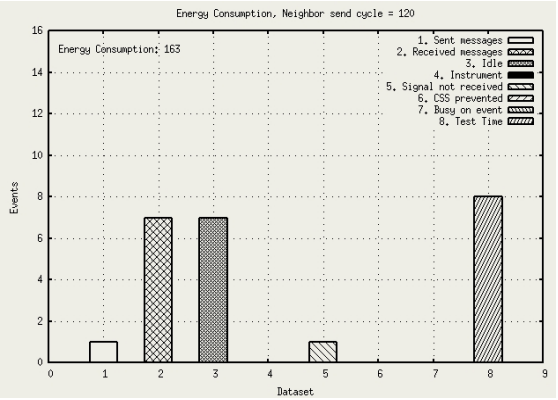
B.3 Changing the neighbor send cycle



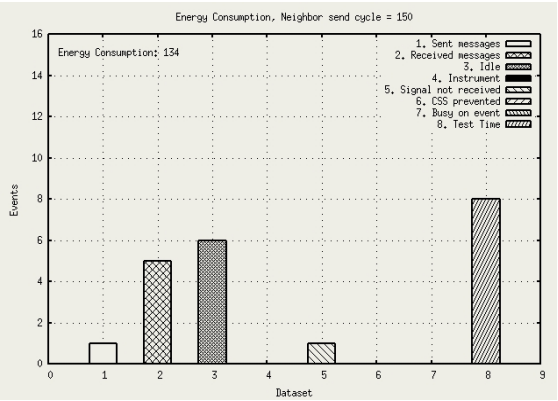
(a) Nice neighbor send cycle = 50



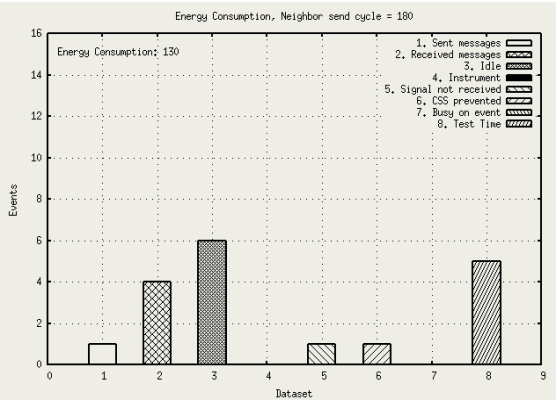
(b) Nice neighbor send cycle = 100



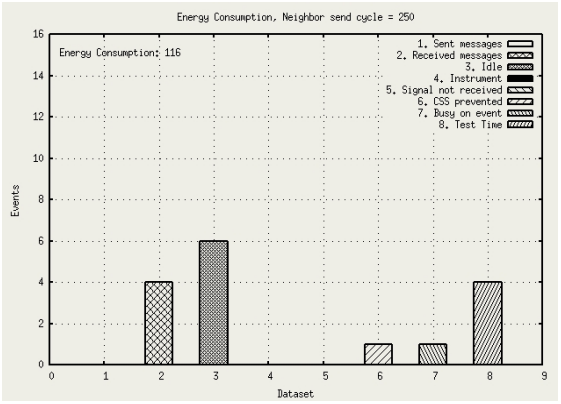
(c) Nice neighbor send cycle = 120



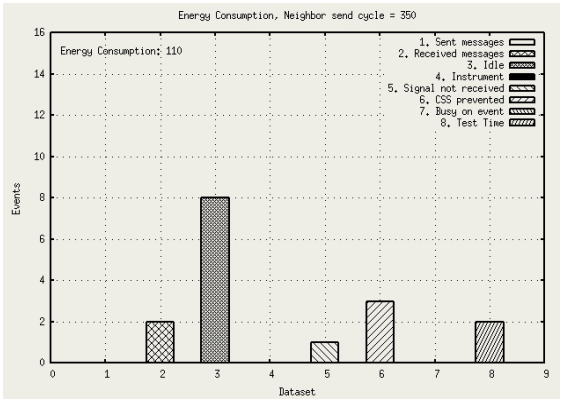
(d) Nice neighbor send cycle = 150



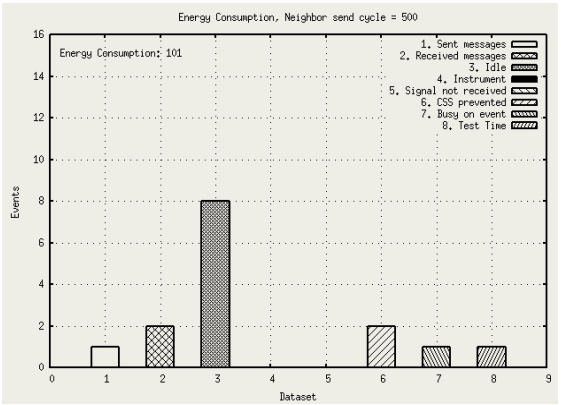
(e) Nice neighbor send cycle = 180



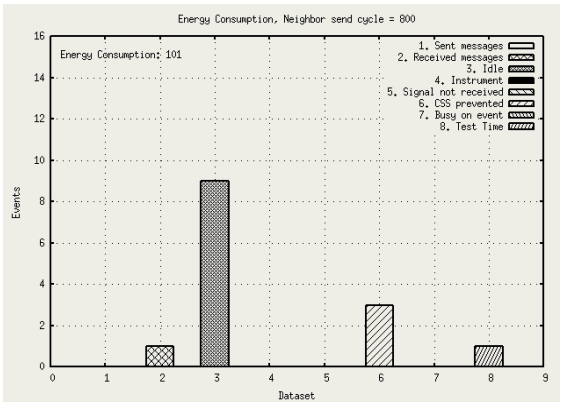
(f) Nice neighbor send cycle = 250



(g) Nice neighbor send cycle = 350



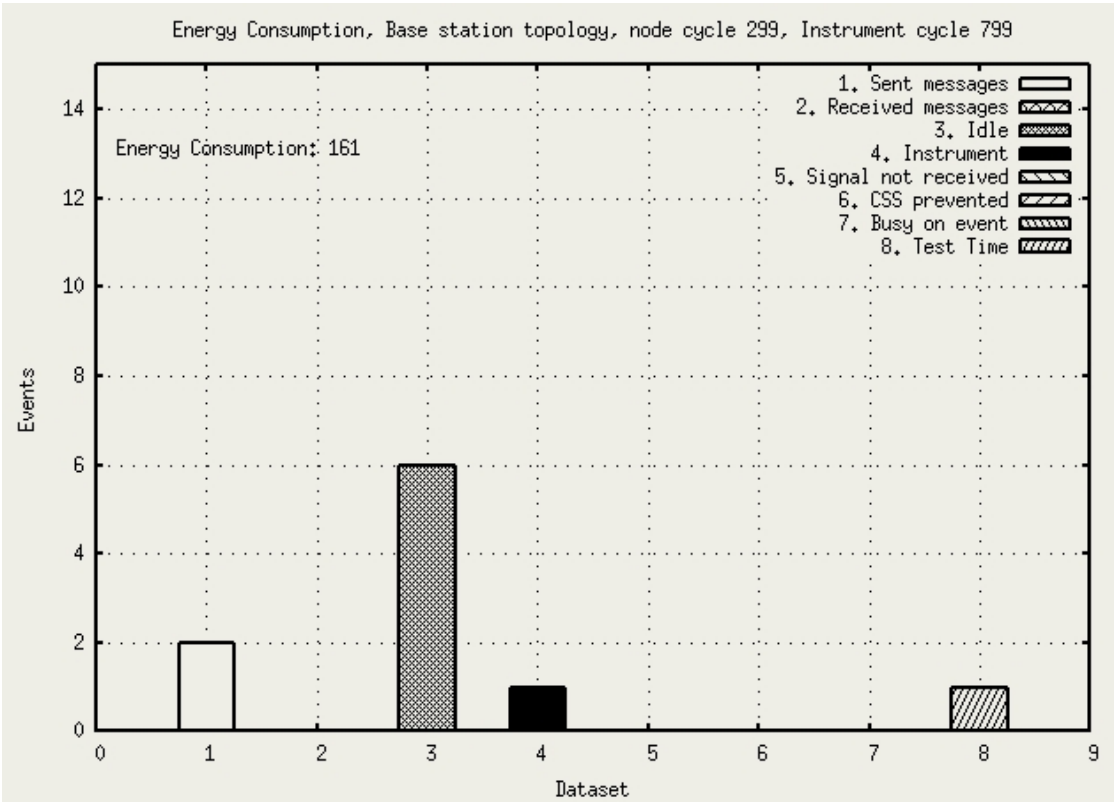
(h) Nice neighbor send cycle = 500



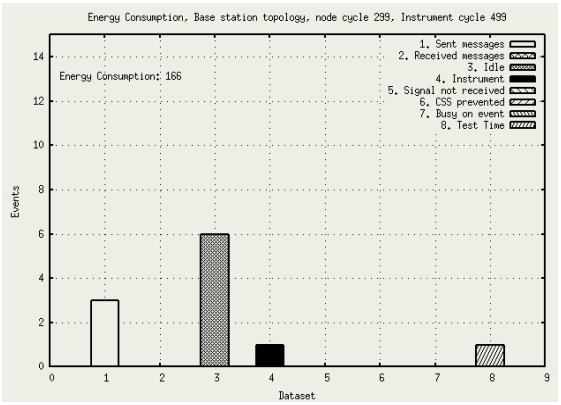
(i) Nice neighbor send cycle = 800

Figure B.3 *Effect of changing the a Nice Neighbor Send Cycle in Line Topology*

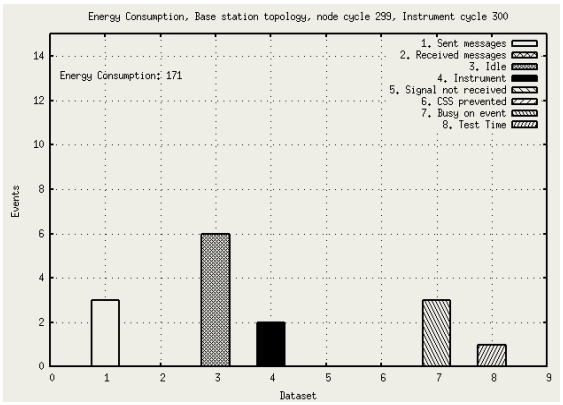
B.4 Changing the Instrument usage cycle



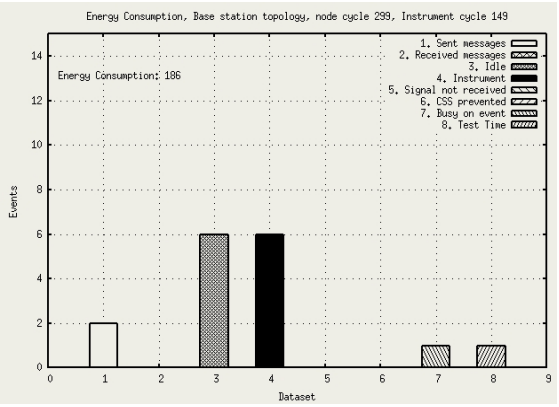
(a) Instrument Usage Cycle = 799



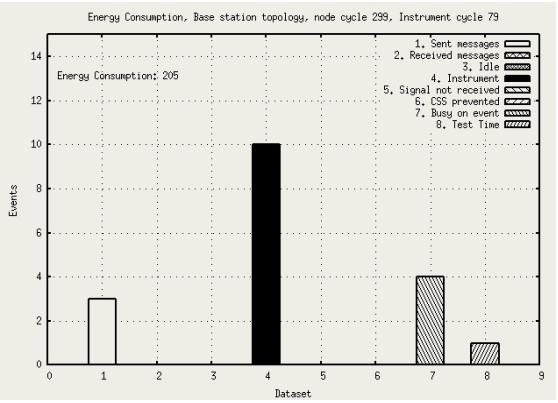
(b) Instrument Usage Cycle = 499



(c) Instrument Usage Cycle = 300



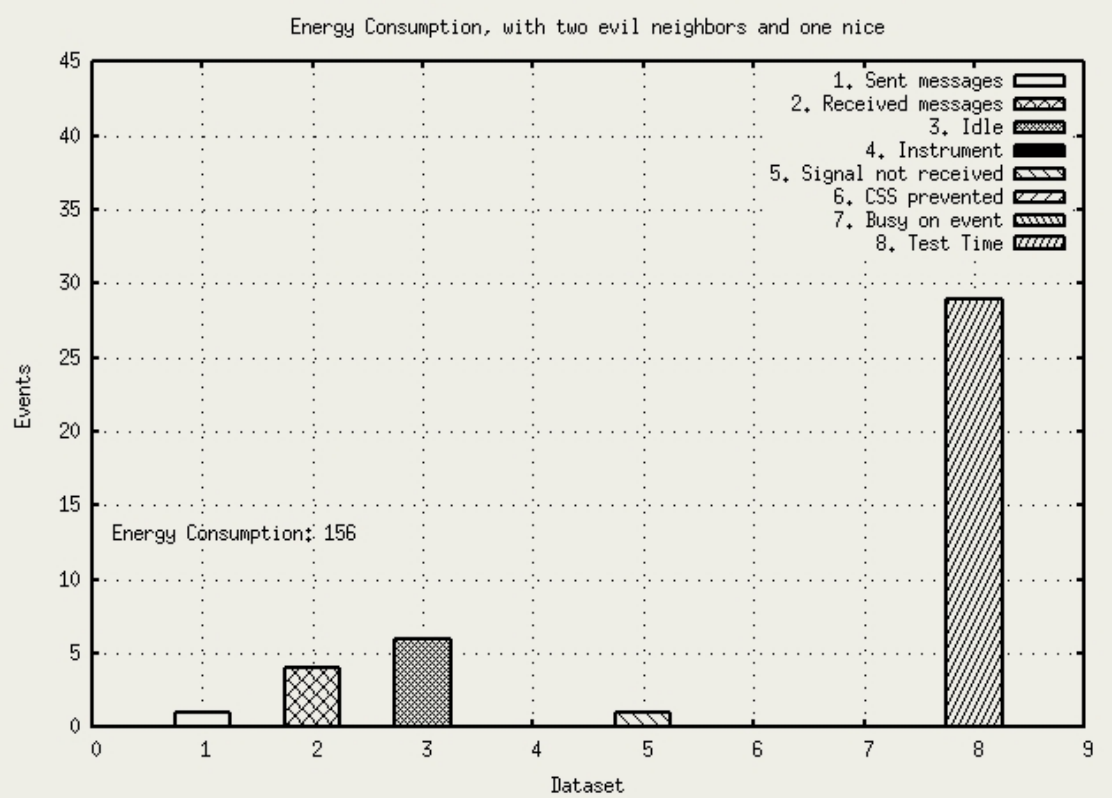
(d) Instrument Usage Cycle = 149



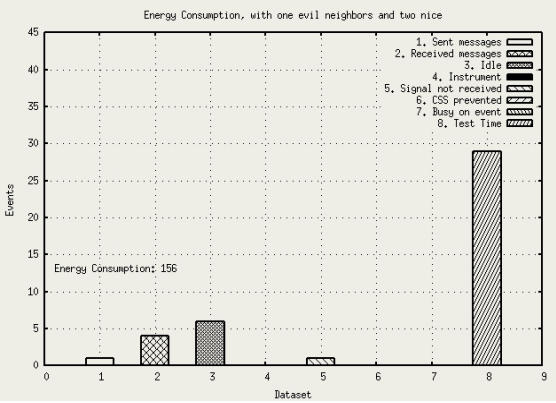
(e) Instrument Usage Cycle = 79

Figure B.4 *Effect of changing the instrument cycle*

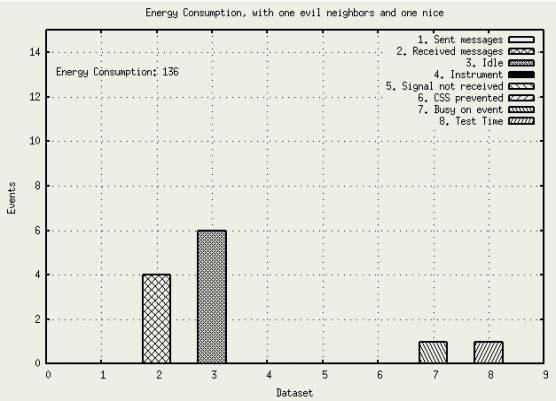
B.5 Topology Test



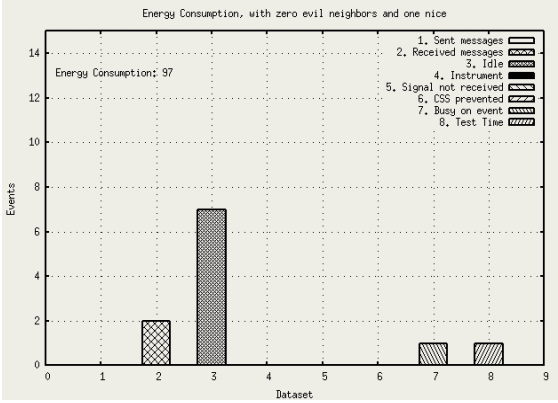
(a) Two evil neighbors, one nice neighbor



(b) One evil neighbor, Two nice neighbors



(c) One evil neighbor, One nice neighbor



(d) Zero evil neighbors, One nice neighbors

APPENDIX C

Model Graphics

C.1 Single Node Delta Protocol

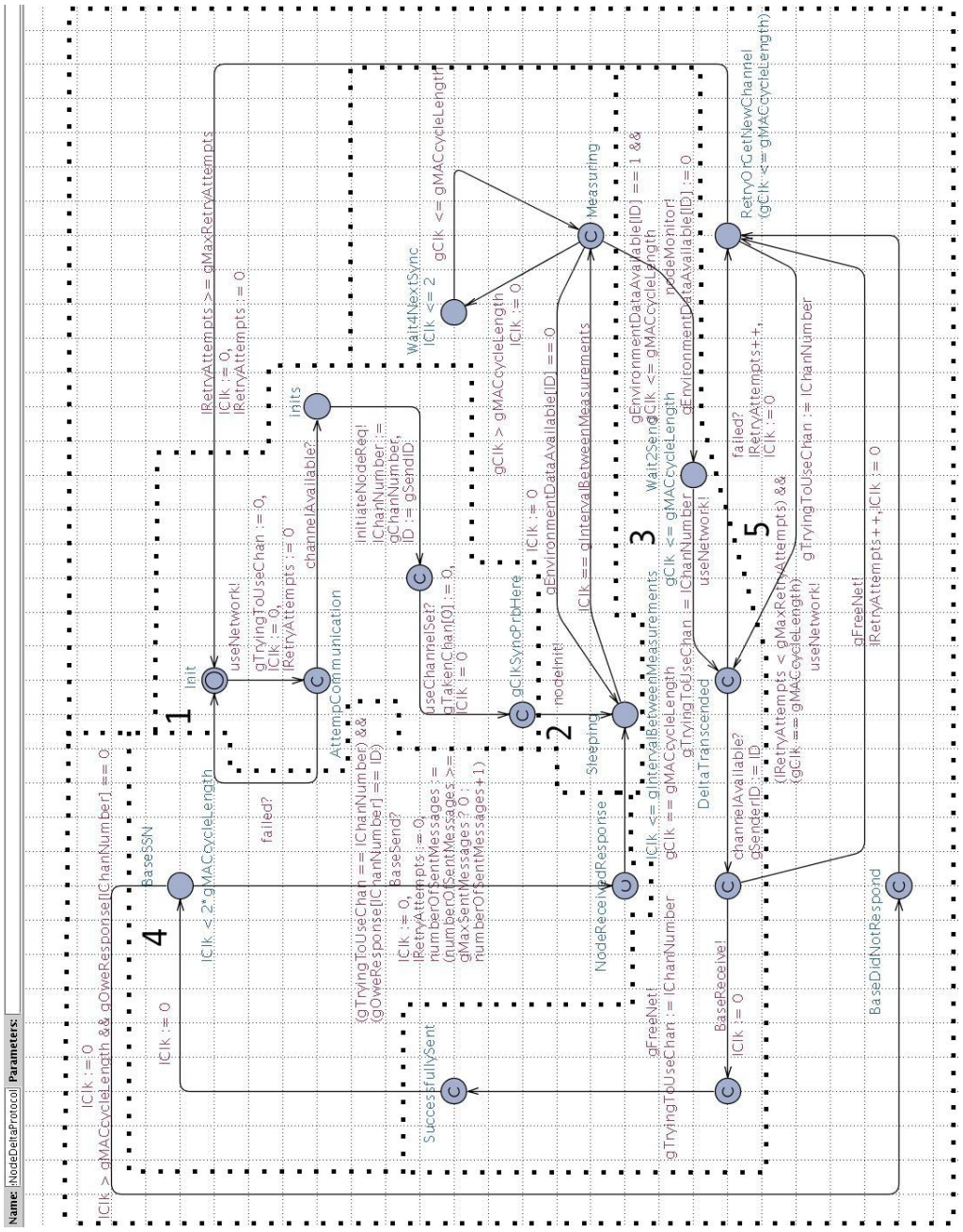


Figure C.1 *Single Node Delta Protocol Model*

C.2 Single Node Time Protocol

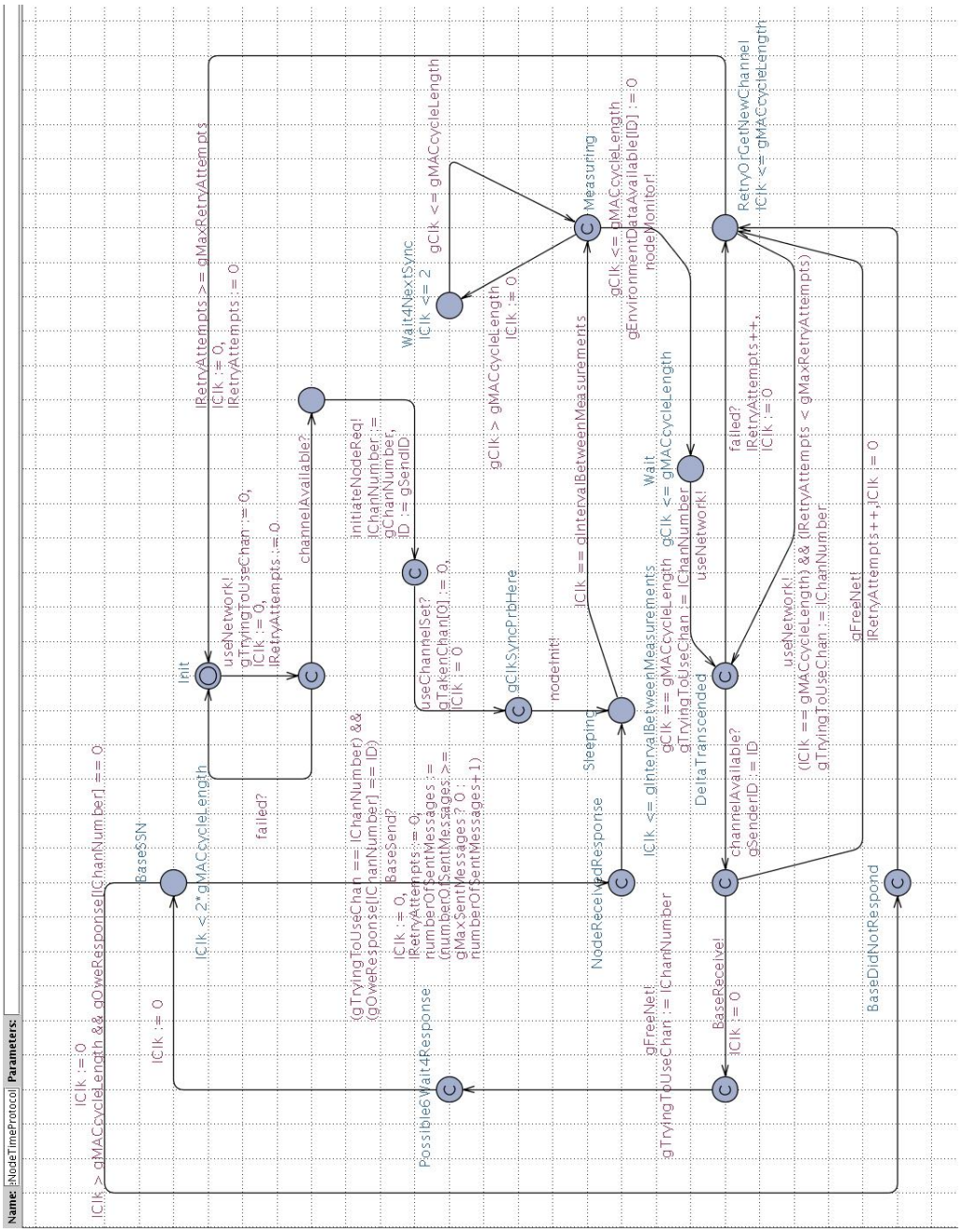


Figure C.2 *Single Node Time Protocol Model*

C.3 Basestation

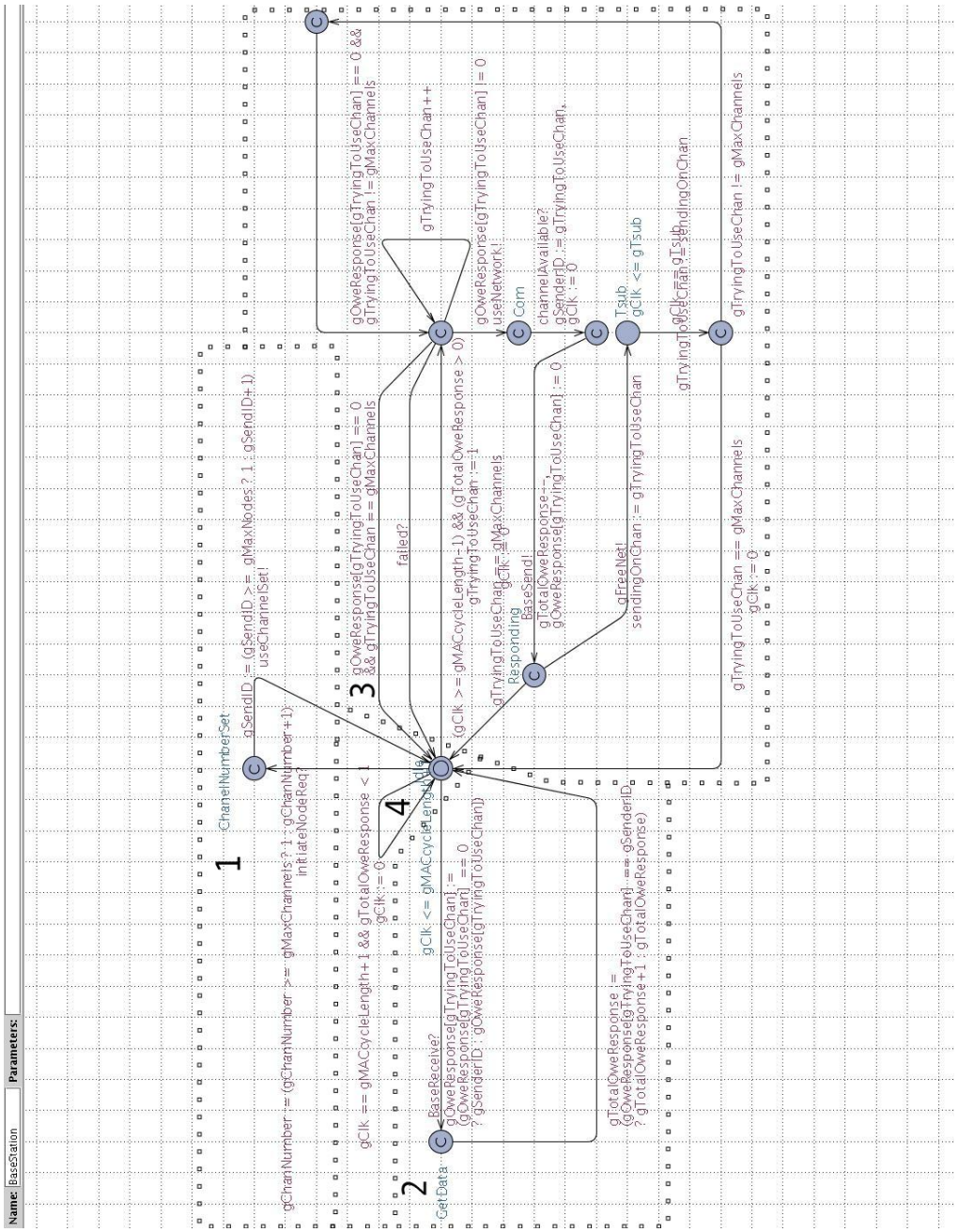


Figure C.3 *Basestation Model*

C.4 Environment

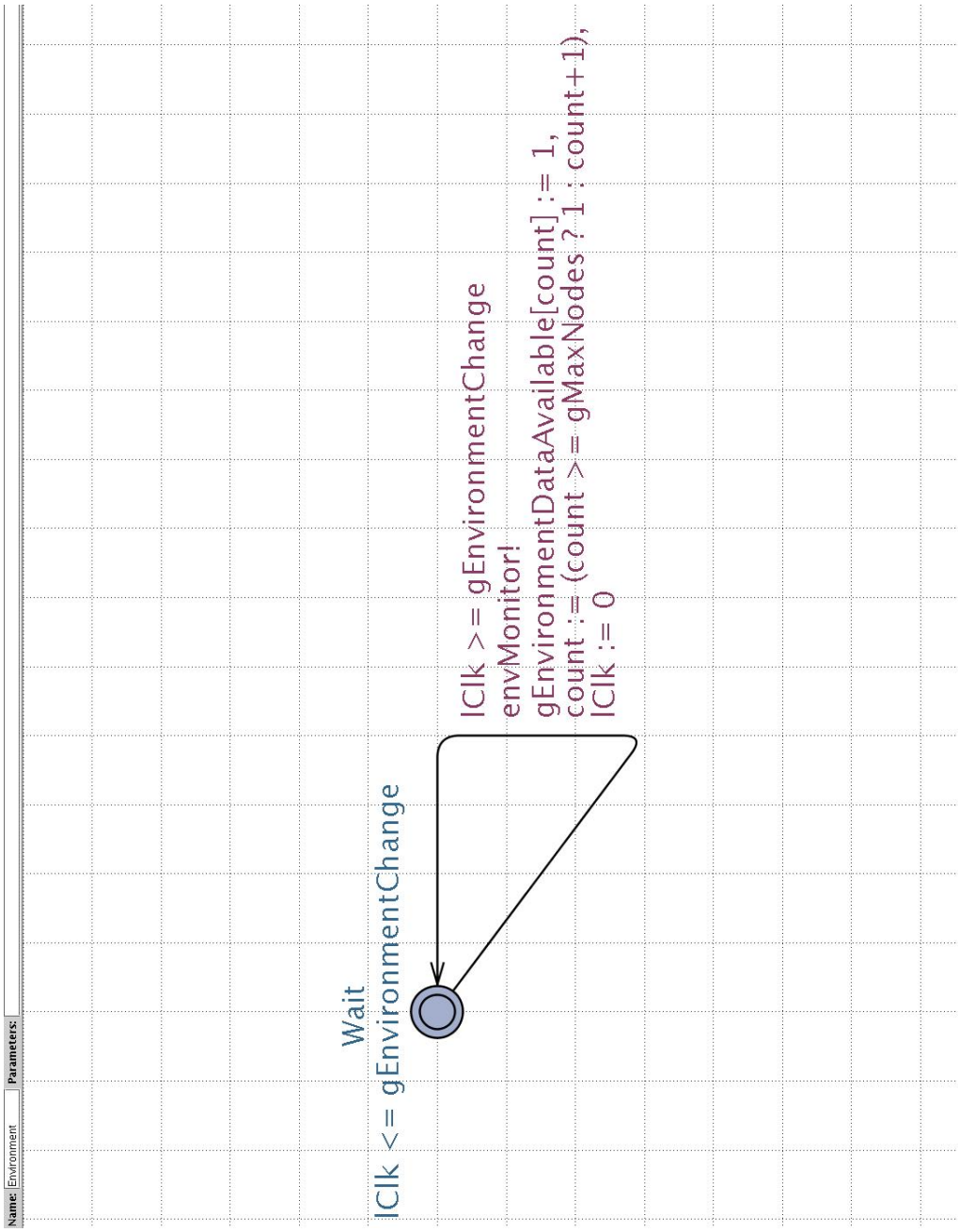


Figure C.4 *Environment Model*

C.5 Network

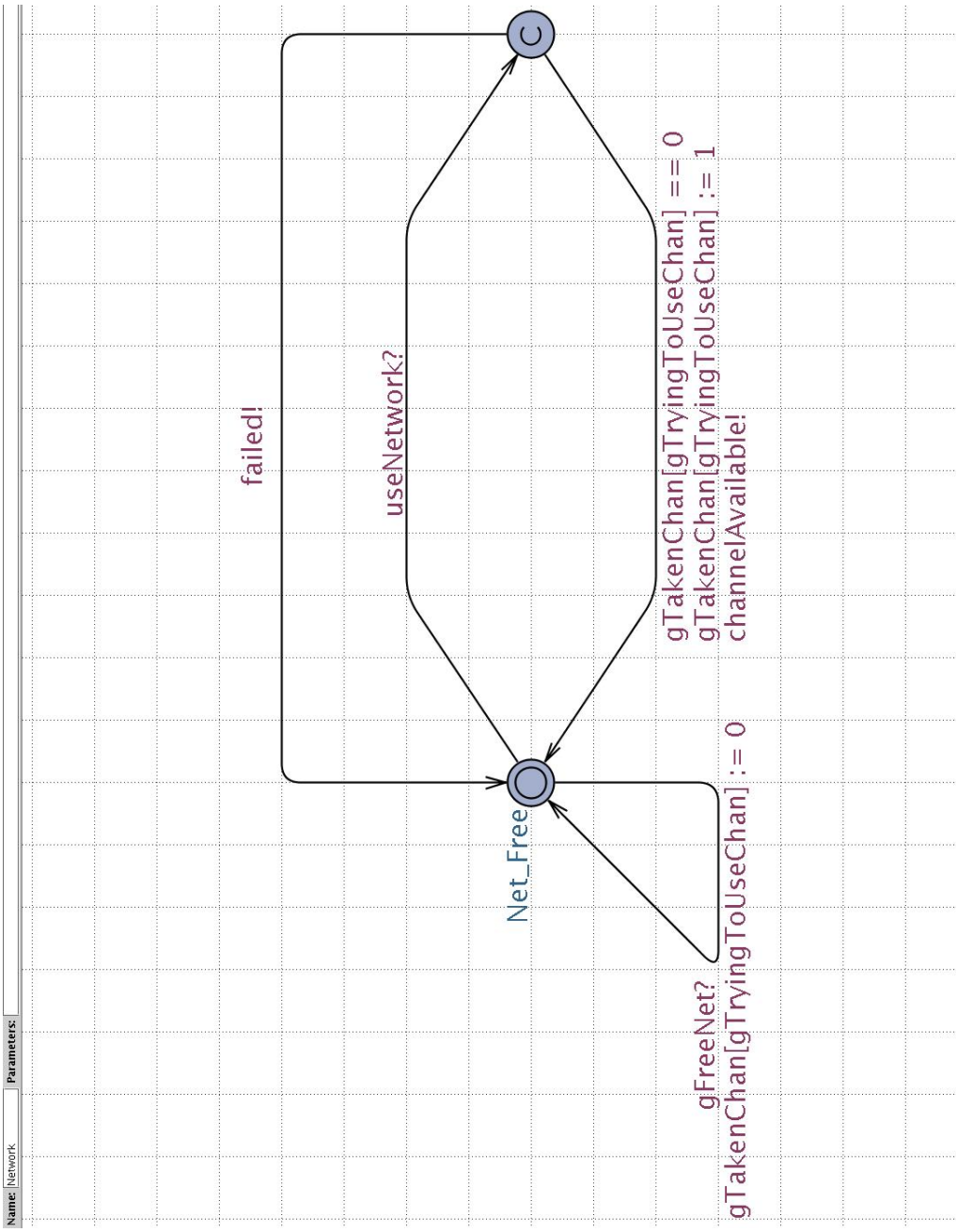


Figure C.5 *Network Model*

C.6 Monitor

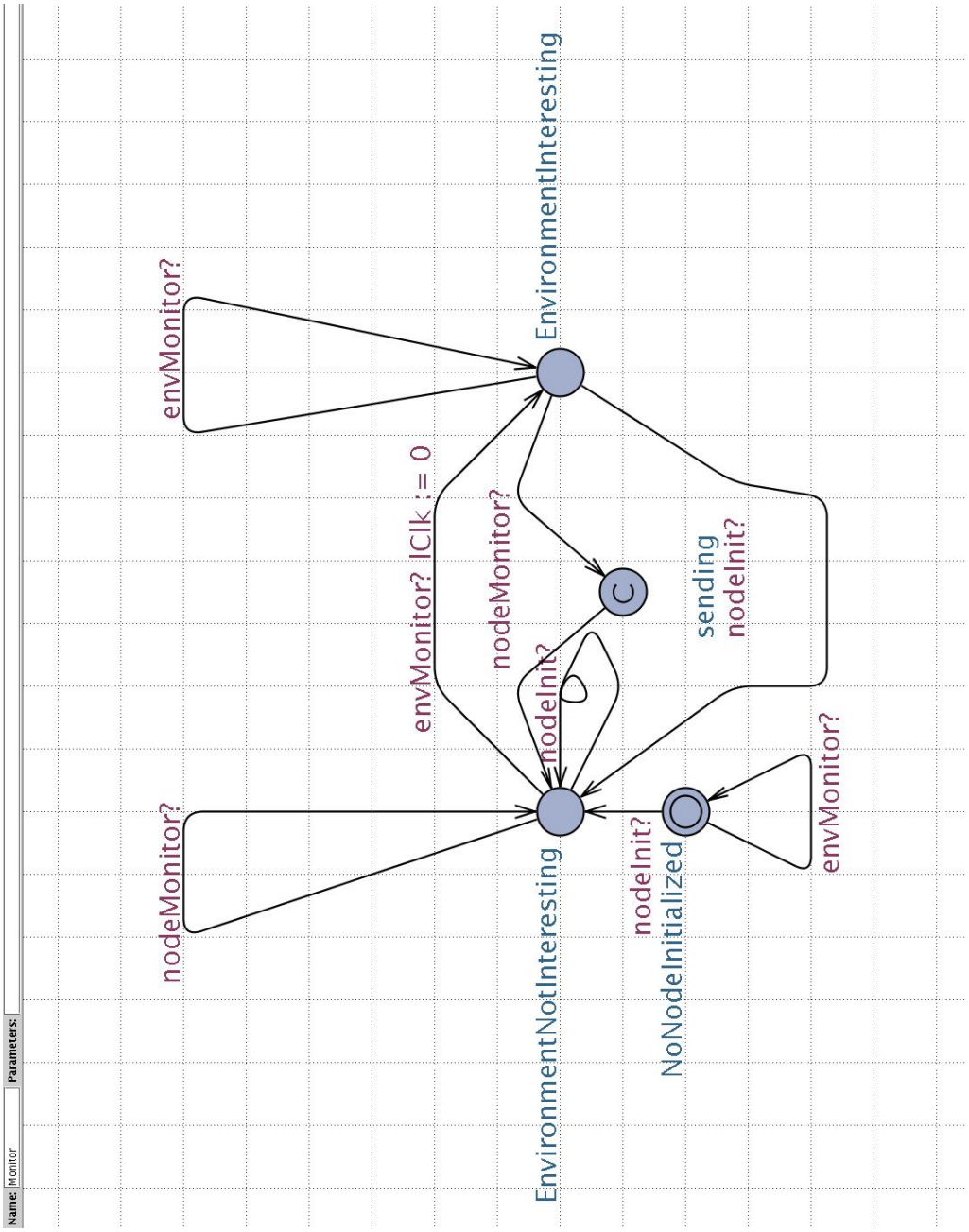


Figure C.6 *Monitor Model*