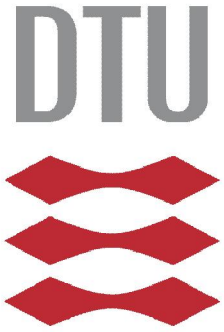


Modelling and Analysis of Wireless Sensor Networks with Energy Harvesting Capabilities

Nan Wu

Kongens Lyngby 2012
IMM-MSC-2012-64

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

A Wireless Sensor Network is a distributed network, where a large number of autonomous sensor nodes are deployed spatially distributed to monitor physical or environmental conditions, and to cooperatively pass their collected data through the wireless channel to a main location.

In design and development of Wireless Sensor Networks, one of the main challenges is achieving long lasting or even infinite lifetime without human intervention. In order to keep the network alive as long as possible, the best energy-wise route need to be found for each data transfer from a source node to the main location. This is made possible through energy-aware routing. Furthermore, energy harvesting is emerging as a solution for a wide range of Wireless Sensor Networks to provide them to last forever.

Though these technologies are encouraging, there are many potential problems exist. One of which is applying these technologies in sensor nodes cost energy itself. There is a risk that the energy required by using energy-aware routing and energy harvesting may exceed the gain by using them. So, it can be seen as a trade-off in Wireless Sensor Networks between risk and return which need to be tested and analyzed.

This Master's thesis provides a generic modelling framework which can be used to model and analyze energy harvesting aware Wireless Sensor Networks. Furthermore, a formal model of this framework is developed in UPPAAL and will be presented in detail. Besides, a Statistical Model Checking (SMC) extension of UPPAAL is applied in the formal model to solve the problem of state-space explosion, and to make the model and the experiments very natural.

With this model in hand, a thorough and formal explanation of energy harvesting aware Wireless Sensor Networks is given. Concrete experiments in terms of simulation and verification, based on the support of UPPAAL-SMC, are also available.

Keywords: WSN, Energy aware routing, Energy harvesting, Timed automata, UPPAAL, UPPAAL-SMC.

Preface

This thesis was prepared at Informatics and Mathematical Modelling at Technical University of Denmark in partial fulfillment of the requirements for acquiring a Master of Science and Engineering.

The thesis deals with modelling and analysis of energy harvesting aware Wireless Sensor Networks.

The project was completed in the period from February 1, 2012 to July 31, 2012 under the supervision of Associate Professor Michael R. Hansen.

Acknowledgement

I would first of all like to thank my supervisor Michael R. Hansen for his support and patient guidance. I have great appreciation for your motivation, assistance and challenges that you have given me through the period of this thesis.

My gratitude also goes to the processors and PhD students of Section Embedded System Engineering at IMM of DTU, who have attended my mid-term presentation. They gave me very inspired suggestions and sincere encouragement.

I also thank my family and friends for their input and support on issues and challenges that I have provided them with the past six months and especially the last couple of weeks.

Finally, a very special thanks to my girlfriend Yuwei Liu for her endless support and help through the entire process of the project leading to this thesis. Also, I need to thank her greatly for her countless hours of proofreading.

Contents

Abstract ..... **i**
Preface ..... **iii**
Acknowledgement ..... **v**
Contents ..... **vii**
CHAPTER 1.
Introduction **1**
1.1 Wireless Sensor Networks 2
1.2 Applications of WSNs 3
1.3 Energy Awareness and Energy Harvesting 5
1.4 Related Work 6
1.5 Problem Formulation 8
1.6 Thesis Structure 9
CHAPTER 2.
A Generic Framework for WSNs **11**
2.1 Wireless Sensor Networks 11
2.2 Sensor Node 12
2.2.1 Component 12
2.2.2 State 13
2.2.3 Event 14
2.2.4 Message 15
2.2.5 cost 16
2.2.6 Operation 17
2.3 Instantiations 19
2.3.1 DEHAR 20
2.3.2 Directed Diffusion 22
CHAPTER 3.
Extensions to the Generic Framework **24**
3.1 Communication Medium 24
3.2 Energy Model 26
3.2.1 Capacitor 27

3.2.2 Battery 28
3.3 Environment Model 29
3.4 Instantiation 33
CHAPTER 4.
UPPAAL Model of WSN Framework 35
4.1 Structure of the WSN model 35
4.2 Components 36
4.2.1 Sensor 36
4.2.2 Transmitter 37
4.2.3 Receiver 39
4.2.4 Processor 40
4.3 Communication Medium 46
4.4 Energy Model 47
4.5 Environment Model 49
4.6 Discussion 51
CHAPTER 5.
Experiments: Simulation and Verification 53
5.1 Preliminary Experiments 53
5.1.1 Simulation 54
5.1.2 Verification 58
5.2 Comprehensive Experiments 59
5.2.1 Simulation 60
5.2.2 Verification 62
5.3 Discussion 64
CHAPTER 6.
Extension 66
CHAPTER 7.
Conclusion 68
References 70
Appendix A
Full UPPAAL Model 73

CHAPTER 1.

Introduction

In recent years, collecting and processing real-time data has been an interesting subject of embedded system design. But, to collect data using traditional wired sensor networks has always been difficult and expensive. The wired network often goes too crowded and creates an obstacle to mobility requirements of the applications. So, Wireless Sensor Network (WSN) has become a premier research topic in the embedded field. WSN is a term used to introduce a class of embedded communication devices that provide reliable wireless connections between sensors, processors and actuators. It is emerging as a solution for a wide range of data gathering and processing applications. WSN has ability to improve people's life but at the same time poses many great system-design challenges. The technological innovations in the area of low-cost/energy-efficient devices, software/hardware co-design and wireless network supporting have been proposed to speed up the development of WSNs.

The purpose of this chapter is to:

- Provide an introduction of Wireless Sensor Network.
- Introduce the concepts of energy awareness and energy harvesting in WSN field.
- Give some related work and practical applications of WSNs.
- Introduce some central problems of energy-efficient WSNs and related approaches to solve them.
- Provide an overview of the thesis structure.

After reading this chapter, the reader will get the basic concepts of Wireless Sensor Networks. Some important issues arising in design and development of WSNs will be introduced. Furthermore, the reason for making a formal model for WSNs will be interpreted. The reader will also obtain the necessary knowledge to understand the following chapters.

1.1 Wireless Sensor Networks

A Wireless Sensor Network is a collaborative network, which consists of spatially distributed sensor nodes (or simply “nodes”) to monitor physical or environmental conditions, such as temperature, light, vibration, motion or pollutants. Typically, a node contains a processor and multiple types of memory, may have a transceiver, include a power source, and accommodate various sensors. These components are integrated on a single or multiple boards, and packaged in a small box. A WSN usually consists of tens to thousands of such nodes that communicate through wireless channels for information routing and processing. A sample network structure is shown in Figure 1. 1.

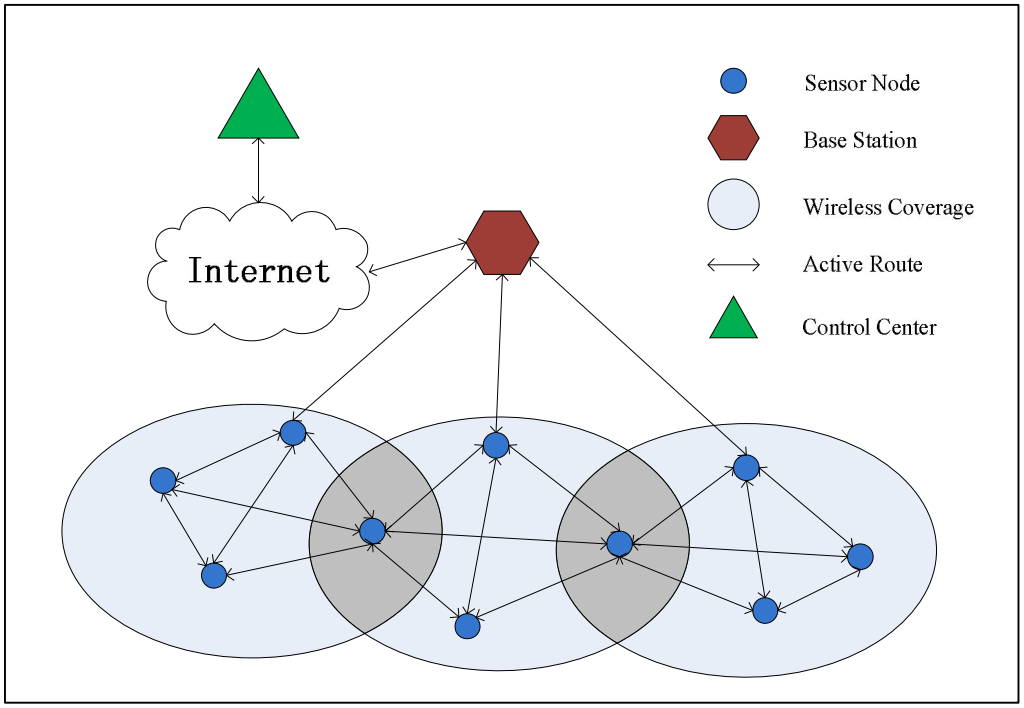


Figure 1. 1 A network structure of WSN

Each node collects information about its environment and cooperatively transfers data through the wireless network to a sink node (e.g. the base station). The collected information is processed either at the node, or at the base station, or in any combination of these. Sensor nodes of energy-efficient WSNs are usually battery-driven, and hence work on an extremely frugal energy budget. They must have a lifetime on the order of months to years, since battery replacement is not an option for networks with thousands of spatially distributed nodes. This makes energy efficiency a particular important concern when designing energy-efficient WSNs.

In a WSN there are two major sources of energy usage:

- Operation of a node, which contains sampling, storing and possibly processing of sensor data.
- Routing data in the network, which includes sending data sampled by the node or receiving and resending data from other nodes in the network [1].

The most substantial challenge faced by designers of small but long lived sensor nodes, is the requirement for extraordinary low-level energy consumption. So, a lot of low-power design techniques have been applied in design of sensor nodes to make them ultra low-power devices, which consume very little energy when operated and even less when being idle. The low-power design techniques emphasize the adjustability of energy consumption with factors such as available resources, event frequency, duty-cycle operation and desired system performance. They help to reduce node energy consumption by exploiting power management in both idle and active state. Thanks to these techniques, long lasting of a WSN is made possible.

The energy dissipated by communication is another key issue in the development of large networks with hundreds to thousands of distributed wireless sensors. An ad-hoc, multi-hop network is assumed for such network since transmitting data directly from the nodes to the base station is not afforded in energy concern. Instead of point-to-point communications, broadcast communication paradigm is mainly applied in order to reduce complexity. Traditionally, low-power, low-range radios are used to route data due to its little energy usage. Because of the vast amounts of data transmission, the duty-cycle of data sending and receiving is a significant factor of energy consumption on broadcast communication.

A lot of research efforts in the field of embedded system, such as low-power design, dynamic power management, and energy-aware routing algorithms have been devoted to achieve lifetime maximization of WSNs. With the advances in these technologies, more and more applications based on WSN are mature and they are becoming an indispensable part of people's life, and will have a great impact on the future.

1.2 Applications of WSNs

Wireless Sensor Networks are application specific, and they have been used in a wide variety of practical applications in recent years. These applications can be categorized into military, environment, medical, agriculture, industry, home and other commercial areas. Some examples of these applications are introduced below.

Wireless Sensor Networks can be used by the military for a number of purposes such as enemy intrusion detection and monitoring enemy activity [2]. The scalable deployment, dynamic topology, self-organization and reliable communication characterizes of WSN make it a very functional technique for military applications. However, there are still some limitations in existing products such as small number of nodes which is necessary to be improved to meet

current military requirements.

Wireless Sensor Networks have been involved in many applications of environment science research. This includes sensing oceans, forests, pollutions, etc. One of the most common instances is air pollution monitoring [3]. The sensor nodes can be deployed in many different locations of a city to monitor the concentration of dangerous gases for citizens. With the advances in WSNs, the environment agency can collect real-time, large scale and comprehensive environmental data for air pollutions in remote location. Another practical environment application is forest fires detection [4]: a network of nodes which are equipped with sensors to measure temperature, humidity and gases can be installed in a forest to detect when a fire happens. Also, they may be equipped with energy harvester like solar cells since the sensors may be left unattended for months and even years. The early awareness of a fire is very crucial for an effective rescue of the firefighters, and therefore a successful protection for both people and environment in time.

An emerging application of Wireless Sensor Networks is patient monitoring system used in healthcare [5]. In a hospital or at home, patients are asked to outfit with wireless, wearable sensors so that doctors, nurses or private medical practitioners can monitor the status of their patients. Placed on the body, various sensors are able to have detection of physiological events such as heart beat, blood pressure, breathe rate and so on. Then these data will be forwarded to a base station, which could be the patient's smart phone. Finally the base station could send them to the doctors' computer via internet. Thanks to the mobile characterize of WSNs, those patients with non-fatal illness may go back home or go to work under their doctors' monitoring.

Wireless Sensor Networks are also used in agriculture area, such as in modern greenhouse. Greenhouse environment monitoring system [6] is essential to improve the productivity through controlling the greenhouse climate. Continuous monitoring of some significant factors for the quality of plant grown such as temperature, light, humidity gives valuable information to the farmer to make better adjustment. Furthermore, compared with cabled measurement used in traditional greenhouse, this wireless monitoring system is cheaper and more flexible to relocate.

Wireless Sensor Networks can be designed and implemented in each type of industry, and a lot of applications can be identified in this area. Different from in other areas, e.g. environmental monitoring, industrial applications have strict requirements in terms of safety, reliability and efficiency. An application to the modern automobile has increased interest recently [7]. The measurement and monitoring of mechanical status such as speed, fuel consumption rate, and tire pressure and illumination intensity offer the driver a safe and efficient automobile journey. Moreover, WSNs allow reducing the weight and volume required by traditional cabling monitoring system. Besides automobile monitoring system, WSNs have been applied in many other industrial applications like automation monitoring and factory process control.

From the proposed examples, we can see that applications based on WSNs in various areas have different requirements. It is obvious that in military safety and reliability are most crucial

properties, and the system performance is considered in the first place. Assuming such a scenario, if a missile detection message is abandoned in order to save energy in military application, the consequence will be very serious. However, in some other areas, e.g. in air pollution monitoring system, it is acceptable to lose pollutants data of some nodes since these loss will not have a significant impact on the air quality measurement of the whole city. So, it is necessary that the developers take the advantage of specific WSNs according to their particular application requirements.

Today, many researchers are currently engaged in developing and implementing more applications based on WSNs. It is certainly possible to extend the classification with more categories. Along with the current researches, WSNs will happen more frequently in people's life in the future.

1.3 Energy Awareness and Energy Harvesting

Energy efficiency is a primary concern in Wireless Sensor Network and it has been widely studied. For a WSN only powered by standard batteries, it is not difficult to imagine that the whole network will lose effectiveness after some critical nodes' batteries gradually drained. The shortest path to base station is implicitly also the least energy consuming path to base station. This path does, however, not consider how much energy is available along this path or any other path [8]. If all of the nodes in the network make use of their shortest path, unavoidably those nodes spatially close to the base station will be involved much more frequently than those far away from the base station, like the nodes in the bottleneck region as shown in Figure 1. 2. As a consequence, these nodes' batteries will be drained very quickly, and then the rest of the network will be cut off from the base station. Therefore, one solution is to find a dynamic routing strategy which takes the energy level of the nodes into account, to avoid the nodes with too little energy, so that continuous lifetime, at least prolonged lifetime of the whole network can be achieved. This strategy is called *energy awareness* and there has been a lot of routing algorithms called *energy-aware routing algorithms* are created based on this strategy.

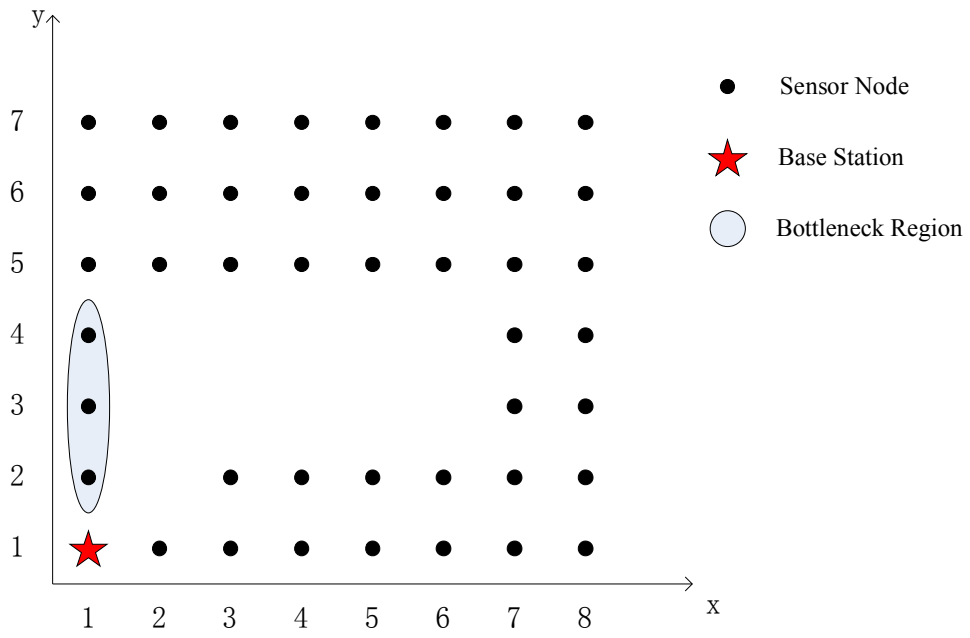


Figure 1. 2 A network structure with illustrating critical nodes

In order to further improve the lifetime of Wireless Sensor Network, there has been another technique called *energy harvesting* being used in each node. Energy harvesting is the process by which energy is obtained from the renewable environmental sources (e.g. solar power, wind energy and thermal energy). Since sensor nodes are usually deployed in harsh environments without human intervention, their lifetime is limited by the life of their batteries. But because the environmental energy is sustainable, the continuous operation of WSN is possibly achieved through energy harvesting. For the battery-operated case, standard batteries can be replaced by rechargeable ones, and with a solar panel as the energy harvester each node can regain "free" energy continuously during daytime and therefore extend the lifetime of the whole network.

1.4 Related Work

Related work in the area of energy harvesting aware routing algorithms can be roughly divided into three classes: *energy efficient*, *residual energy aware*, and *energy harvesting aware algorithms*.

To address the problem of static shortest path algorithm, energy efficient algorithms [9, 10, 11] have been proposed. The aim of these approaches is to increase the lifetime of the network as a whole by distributing the routed data to several neighbours in order to minimize the energy consumption of the nodes on the shortest path. However, these approaches do not measure the residual energy in the batteries.

The residual energy aware algorithms [11-18] are all measuring the residual battery energy and are extending the energy efficient algorithms to take the available energy of the nodes into account. These approaches have a common assumption that the residual battery energy is monotonically decreasing. However, it is not a realistic assumption for most networks and it does not accommodate for energy harvesting since energy increasing cannot be handled.

The energy harvesting aware algorithms do not assume monotonically decreasing battery energy. Furthermore, they may estimate the future harvested energy to prolong the lifetime of each node. Thanks to energy harvesting technologies, it is possible to eventually run the WSN without batteries. However, using adaptive energy management at the node level adds to the complexity to the WSN system. One of the trickiest issues is that the amount of future harvested energy is not predictable. For instance, the amount of energy being harvested by a solar panel over a period of time varies according to different conditions, such as the time of a day, local weather, and shadows from buildings, trees, etc. Some energy harvesting aware algorithms like [19-24] have been established to address these problems and they can accommodate for both decreasing and increasing of battery energy. However, these routing algorithms do not consider dynamic routes for multi-hop networks.

Existing techniques [25-30] for managing harvested energy in WSNs are mostly focusing on energy management at the node level. It means that the energy being harvested on a node can be only used on this node's operation. Therefore these techniques cannot make network wide decisions and do not have a helpful impact on energy-wise routing.

An algorithm using harvested energy in its routing is [19], and it does this in a very simple but not natural way. By detecting whether a node is harvesting or not, the algorithm decides if the node should be selected as the next stop on the path. Obviously it is not reasonable since it does not take residual energy and the estimation of future harvested energy into account.

A mathematical framework for energy harvesting aware routing in multi-hop WSNs have been established in [23]. An algorithm based on this framework is able to use harvested energy to make network wide decisions. The advantage of this framework is that the best energy-wise route for data transmission can be found. However, this approach makes an ideal assumption that changes in node energy level are broadcasted instantaneously to all other nodes. This is neither realistic due to limitations in radios' ranges nor desirable since it would consume a lot of energy to communicate with the whole network. Another problem is that global knowledge of the network is assumed for each node in this approach.

This thesis presents a generic modeling framework for a broad range of energy harvesting aware WSNs [1]. An algorithm called DEHAR [8] which is based on this generic framework is also introduced. It is an approach in which changes in node energy level are broadcasted to neighbour nodes only. The energy deficit will be considered as a penalty for a node distance to the base station. This penalty is dynamically recomputed on the basis of changed energy levels caused by energy consumptions and energy harvesting.

1.5 Problem Formulation

In recent years, the scale of Wireless Sensor Networks applied in various applications is usually large, and hence it is critical that they are designed and implemented correctly. It is unacceptable that people find a system working undesirably after deploying thousands of spatially distributed nodes. Correction and replacement in such case will be very difficult and expensive. With these issues in mind, modelling and analysis a WSN in advance seems an essential and sensible choice. That is because by simulation and verification of a model, many properties of the target system can be known and analyzed before implementation and deployment.

Since Wireless Sensor Networks are usually applied in applications which need to live a relatively long period of time in harsh environments without infrastructure power supply, energy efficiency perhaps becomes premier research topic in this field. For energy-efficient WSNs, the balance between energy consumption and system performance is one of the most important properties need to be considered. For example, at the hardware level, the energy consumption characteristics are affected by the duty-cycle operation of a sensor node. It is implicitly that a transceiver will not lose any messages but simultaneously consume a lot of energy if it keeps active all along. As WSNs are application-specific, this trade-off varies depending on the requirements of specific applications. In military area, for instance, correct and complete transmission of messages is the first objective, and hence keeping the sensors and transceivers always working seems a judicious choice. However, in some other areas, like environmental monitoring, it is acceptable to lose some messages due to applying duty-cycle operation in sensors and transceivers. As a consequence, a little energy can be saved and therefore the lifetime of the network will be prolonged. So, with the model of a WSN in hand, researchers can adjust its structure and configuration according to the requirements of specific applications during early design stage, and check or analyze concerned properties of the target system.

Over the past decades a variety of methods and tools have been developed for checking models of hardware and software designs. A lot of approaches can be chose to deal with WSN case. One of the most common approaches is *Communicational Sequential Processes* (CSP), which is a formal modelling language for describing patterns of interaction in concurrent systems. There have been a number of tools for analyzing and understanding systems described using CSP over the years and they are often described as model checkers, such as FDR2. But due to the characteristics of WSNs are connected to timing and synchronization, such as event frequency, communication between nodes and synchronization between different components of a node, *timed automata* [31] seems a better choice to model such system. A timed automaton is a finite automaton extended with a finite set of real-valued clocks. It can be used to model and analyze the timing behavior of real-time systems or networks. There

exist a number of tools to input and check timed automata, including the model checker *UPPAAL* [32]. UPPAAL is an integrated environment for modelling, simulation and verification of real-time system. In this Master project, it is selected as the model checker to build and verify a Wireless Sensor Network model. In the following, the reader is assumed to have the general knowledge of timed automata and UPPAAL.

Due to the large scale feature of usual WSNs, a problem arises in the process of its model checking. The more nodes are created in the UPPAAL model, the more parallel states and undecidable problems need to be handled with in verification. Unfortunately, the classical UPPAAL engine is not capable of handling case studies because of issues of state-space explosion. Therefore, a SMC extension of classical UPPAAL, where SMC represents *Statistical Model Checking*, needs to be introduced in this Master thesis to solve these problems.

Statistical Model Checking refers to a technique that monitors finitely many runs of the system and then uses a statistical algorithm to obtain an estimate for the system. The core idea of SMC is to monitor a number of simulations of a system whose behaviors depend on a stochastic semantic, and then uses the results to get an overall estimate of the probability that the system will behave in some manner [33]. It can be seen as a trade-off between test and formal verification.

Although such simulation-based approach cannot guarantee a result with 100% confidence, it allows a user to specify complex problems in the form of probability distributions and compare probabilities to analyze system performance. Furthermore, it is possible to bound the probability with time or cost. The UPPAAL-SMC engine has the following typical queries:

- A qualitative check: **Pr [time <= bound] (< > expr) >= p.**
- A quantitative check: **Pr [time <= bound] (< > expr) ?.**
- A comparison check :
  Pr[time1 <= bound1] (< > expr1) >= Pr[time2 <= bound2] (< > expr2).

All these checks rely on some statistic algorithms to estimate the correctness by observing simulations of the system. These UPPAAL-SMC queries allow us to verify properties that cannot be expressed in classical temporal logics and to approximate undecidable problems with probability.

1.6 Thesis Structure

Generally speaking, the purpose of this Master thesis is to:

- Provide an introduction of Wireless Sensor Network.

- Highlight some important concepts in energy-efficient WSN area, which includes energy awareness and energy harvesting.
- Introduce a generic modelling framework for energy harvesting aware wireless sensor networks.
- Give some natural extensions of the generic modelling framework.
- Establish a formal model of the generic framework and its extension in UPPAAL with the support of SMC.
- Present some simulations and verifications of the UPPAAL model.

The thesis is structured as follows:

- Chapter 2 introduces a generic modelling framework which can be used to model and analyze a broad range of energy harvesting aware WSNs.
- Chapter 3 gives the extensions of the generic modelling framework in terms of communication medium, energy model and environment model.
- Chapter 4 provides the formal model, with timed automaton refers to each component presented in the generic framework and its extension.
- Chapter 5 presents some simulation and verification experiments in connection with the developed UPPAAL model.
- Chapter 6 offers a brief introduction of an example low-level protocol applied in energy-efficient WSNs, and as an extension chapter it explains the theoretical basis of some assumptions made during the developing of the UPPAAL model.
- Chapter 7 contains a summary and concluding remarks, it also gives some possible directions for improvements and extensions.
- Appendix A provides the full version of the UPPAAL model for the example given in Chapter 4.

CHAPTER 2.

A Generic Framework for WSNs

This chapter presents a generic modelling framework which can be used to model and analyze energy harvesting aware WSNs. This part of the thesis is very faithful to paper [1], and it is introduced here since the generic framework is a conceptual and operational fundament for building the formal model of such networks. Furthermore, two instantiations based on this modelling framework will be given in this chapter.

2.1 Wireless Sensor Networks

The purpose of establishing a generic framework is to have the basis of modelling and analyzing a wide range of energy harvesting aware WSNs with similar characteristics.

For the WSNs concerned in the generic framework, there are some assumptions:

- Sensor nodes are low-cost devices with limited energy source.
- Sensor nodes are equipped with low-power, low-range radios as communication tools.
- The only additional energy supply of a node is through energy harvesting from the renewable environmental sources.
- The routing in the network adapts to the dynamic changes of available energy in the individual nodes, i.e. the routing is energy aware [1].

On the other hand, there are no any particular assumptions about the kind of sensors to presume the type of physical or environmental conditions to be monitored. In addition, there may be more than one kind of sensor integrated in a physical node.

These assumptions lead to a few consequences concerning the working mode of the network. These consequences are shown as follows:

- The wireless sensor network is a multi-hop network since for most of nodes transmitting data directly to the base station is not afforded in energy concern.
- A sensor node may only be able to have a direct communication with a small subset of the other nodes, called its neighbours, due to the range of radio signals.
- A sensor node needs state information about its neighbours reflecting their available energy levels in order to support energy-aware routing.
- A sensor node can make immediate changes to its own state; but it can only affect its neighbours by use of radio communication [1].

These assumptions and consequences ensure that the wireless sensor networks constructed based on this framework will have energy harvesting capabilities and adapt to energy awareness.

The generic framework of energy harvesting aware WSNs is based on the existence of a definition of sensor nodes, a description of the communication medium through which the nodes communicate, and a specification of the environment conditions which affects the energy harvesting capabilities. Only the first issue is considered in paper [1] and it will be introduced in next section. The last two parts will be given as the extension of the framework in Chapter 3.

2.2 Sensor Node

In this section the concrete information of a sensor node in this generic modelling framework will be introduced. A sensor node usually possesses a unique identity (*id*), and contains a few physical components. Each node has several states which express its current status including physical condition, energy level and environment knowledge. Besides, some events may occur during the normal operation of a node, and due to the happening of these events a node may need to communicate messages with its neighbours for some reasons. In the following, these matters of sensor nodes will be described in detail.

2.2.1 Component

A sensor node usually comprises five physical *components*:

- A *processor* which is used to control other devices, to implement the energy-aware

routing algorithm, and to react to events and treat messages.

- A *sensor* which is used to monitor physical or environmental conditions.
- A *transmitter* which is used to broadcast messages to the network by radio.
- A *receiver* which is used to receive broadcasting messages from the network.
- An *energy harvester* which can collect sustainable environmental energy, e.g. a solar panel. An instance of energy harvesting device will be introduced in Chapter 3.

It is not difficult to see that the use of processor, sensor, transmitter and receiver consume energy and that the only supply of energy comes from the node's energy harvester.

2.2.2 State

The *state* of a node is partitioned into a *physical state* and a *computational state* [1]. The specific content of the states is described as follows:

- PhysicalState(*ps*) – which models the energy level of a battery (Energy(*e*)) as well as the dynamic changes of the energy harvesting devices, such as a capacitor.
- ComputationalState(*cs*) – which models the state in the processor, including approximation about the energy level and knowledge of the neighbours, and also a description of the communication medium, which includes, for example, a neighbourhood area of the node.
- AbstractState(*as*) – which is a simple version of a computational state and it is used to be communicated to neighbour nodes. It is introduced since it will cost too much energy if communicating complete computational state to neighbours.

These states are used to support energy-aware routing and they are supposed to be changed by a number of *state functions*, by which a node is kept dynamically aware of its energy optimized next destination (or simply "next destination"), during the operation of the processor. A concrete definition of these state functions is given in Figure 2. 1:

- consistent?(*cs*) is a predicate which is true if the computational state *cs* is *consistent*. Since neighbour and energy information, which are used to guide the routing, are changing dynamically, a node may end up in a situation where no neighbour seems feasible as the next destination on the route to the base station. Such a situation is called inconsistent, and the predicate consistent?(*cs*) can test for the occurences of such situations.

- abstractView(*cs*) gives the abstract view of the computational state *cs*. This abstract view constitutes the part of the state which is communicated to neighbours.

- updateEnergyState(*cs, e*) gives the computational state obtained from *cs* by incorporation of the actual energy level *e*. The resulting computational state may be inconsistent.

- updateNeighbourView(*cs, id, as*) gives the computational state obtained from *cs* by updating the neighbour knowledge so that *as* becomes the abstract state of the neighbour node N_{id}. The resulting computational state may be inconsistent.

- updateRoutingState(*cs*) gives the computational state obtained from *cs* by updating the routing information on the basis of the energy and neighbour knowledge in *cs* so that the resulting state is consistent.

- transmitChange?(*cs, cs'*) is a predicate which is true if the difference between the two computational states are so significant that the abstract view of the "new state" should be communicated to the neighbours.

- next(*cs*) gives, on the basis of the computational state *cs*, the identifier of the "best" neighbour to which observations should be transmitted.

Extracted from -- section 2. A Generic Modelling Framework in [1]

Figure 2. 1 The definition of state functions

For the function consistent?(*cs*), "no neighbour seems feasible as the next destination" means that a node has the shortest distance to the base station compared to its neighbours, but it is still not able to transmit data directly to the base station. The distance mentioned here is defined as *energy-adjusted distance*, which incorporates that the energy deficit is converted as a penalty on physical distance, in order to take the available energy level into account in routing strategy.

The last function transmitChange?(*cs, cs'*) is simple yet essential, as only significant energy changes in a node need to be notified to its neighbours. It is neither feasible nor desirable to broadcast minimal changes to the network, since it requires too much communication and thus too much energy is spent on this administrative issue compared to the energy used for sensing and routing environment observations.

2.2.3 Event

The processor in a node can react to *events* originating from sensor observations, messages

acceptance and dynamic changes of available energy. Let Observation denote the set of environment information and Message denote the set of communication messages. The events are listed in Figure 2. 2:

- observationEvent(*o, ps*), where *o* ∈ Observation is a recorded sensor observation and *ps* ∈ PhysicalState is a physical state which incorporates the energy consumption due to the activation of the sensor.
- receiveEvent(*m, ps*), where *ps* ∈ PhysicalState and *m* ∈ Message, which could be an observation to be transmitted to the base station or a message describing the state of a neighbour node.
- physicalStateEvent(*e, ps*), where *ps* ∈ PhysicalState is a new physical state. This event occurs when a change in the physical state is recorded. This change may, for example, be due to energy harvesting, due to a drop in energy level, or due to some other change which could be the elapse of time.
- readEnergyEvent(*e*, *ps*), where *e* ∈ Energy and *ps* ∈PhysicalState, which is an event signaling a reading *e* of the energy level in the node and a resulting physical state *ps*, which incorporates that the reading actually consumes some energy.

Extracted from -- section 2. A Generic Modelling Framework in [1]

Figure 2. 2 The definition of events

Both physicalStateEvent(*e, ps*) and readEnergyEvent(*e*, *ps*) provided in above figure are energy related events and they are initiated by processor of a node. The difference between them is that whether the event itself consumes energy. The latter one needs to be triggered at appropriate time to investigate if the energy variance of the node is significant, and this investigation actually consumes energy. In the former case, the change of the energy level, for example due to energy consumption of sensor and receiver, is a relatively frequent and cheap operation which does not consume energy. The events observationEvent(*o, ps*) and receiveEvent(*m, ps*) are not related to energy and are passive to processor, they are originated by sensor and receiver respectively.

2.2.4 Message

The *messages* communicated between nodes are divided into *Observation Message* and *Neighbour Message*. The observation message contains the environment information sampled by a sensor and need to be forwarded to the base station. The neighbour message contains the abstract state of the source node. Figure 2. 3 depicts the detail form of these two kinds of messages.

- *Observation Messages* of the form obsMsg(*dst, o),* where *dst* is the identity of the next destination of the observation $o \in$ Observation on the route to the base station.
- *Neighbour Messages* of the form neighbourMsg(*src, as*), where *src* is the identity of the source, i.e. the node which have sent this message, and $as \in$ AbstractState is the contents of the message in the form of an abstract state.

Extracted from -- section 2. A Generic Modelling Framework in [1]

Figure 2. 3 The definition of messages

Since these messages are transmitted in a broadcast way, an observation message needs to contain the identity of its expected destination (*dst*) so that the destination node is able to recognize and accept this message while the other nodes can ignore this message if they receive it. Similarly, in a neighbour message, an identity of the source node (*src*) is required to inform the receivers of this message that where this message comes from, i.e. whose abstract view should be updated.

Each sensor node maintains an input-queue for storing received messages. When a message $m \in$ Message is received from the network, *m* is put into the queue and a relevant event receiveEvent(*m, ps*) will be originated if *m* is the first element in input-queue. Reacting to this event will remove *m* from the queue and a new receive event will be offered as long as there are messages in input-queue. Thus, the received messages waiting in input-queue will be dealt with by processor in the order of receiving of them.

On the other hand, each node has an output-queue for saving messages that need to be transmitted. The transmitter can broadcast the message which is in front of the queue for a period of time. Ideally, nodes which are within the range of the radio signal will receive this message. After the sending interval, this message will be removed from the queue and a new message will be transmitted as long as the output-queue is not empty. Therefore, the messages waiting in output-queue will be forwarded in order.

2.2.5 cost

In a sensor node, the operation of each physical component except energy harvester costs energy. It is feasible to assume that every time the sensor, transmitter or receiver is used, respectively a constant amount of energy is spent on activation and running of the device. Only the energy consumption caused by processor is not certain, since the number of state functions executed on the processor depends on the occurrence of the events described in Subsection 2.2.3. These state functions have been introduced in Figure 2. 1, and for simplicity it is assumed that the cost of executing the predicates consistent? and transmitChange? can be neglected and the execution of each of the other state functions have a constant energy consumption. Therefore, these energy consumptions can be specified in the form of constant

cost functions. These cost functions and their initiator are summarized in Table 2. 1.

Cost functions	Initiator
costSense	Sensor
costSend	Transmitter
costReceive	Receiver
costRead	Processor
costAbstractView	
costUpdateEnergyState	
costUpdateNeighbourView	
costNext	
The cost of the predicates consistent? and transmitChange? are assumed negligible.	

Table 2. 1 A summary of cost functions

The execution of each of the above cost functions will lead a physicalStateEvent and therefore cause a change of the energy level in physical state. It is obvious that the energy consumed by the first three functions is used for collecting and routing data, and by the last five is spent on supporting energy-aware routing. Thus, there is a risk that the energy required by applying energy-aware routing may exceed the gain by using it. So, it is a key problem to design an effective energy-aware routing algorithm which spends reasonable energy on administrative issue so that in general the routing algorithm helps stretch the lifetime of the network as a whole.

2.2.6 Operation

The processor of a sensor node has two main *phases* during its operation. One is *idle* and the other is *treat-message*. In idle phase the processor is in sleeping state at most of time and waits for some events to happen. Once an event is originated, the processor will react to it and sometimes need to make a transition to treat-message phase. In treat-message phase the processor handles a single message and returns to idle phase as soon as it finishes treating the message.

The state changes and message transmission for idle phase are given in Figure 2. 4. The node stays inactive in the idle phase until an event occurs.

```
Idle_id (cs,ps) =
    wait
        physicalStateEvent(ps') → Idle_id(cs, ps')

        readEnergyEvent(e, ps') →
                let cs' = updateRoutingState(updateEnergyState(cs, e))
                let ps'' = costUpdateEnergyState(costUpdateRoutingState(ps'))
                if transmitChange?(cs, cs')
                then let m = neighbourMsg(id,abstractView(cs'))
                        send_id(m); Idle_id(cs', costSend(costAbstractView(ps'' ),m))
                else Idle_id(cs, ps'')

        observationEvent(o, ps') →
                let dst = next(cs)
                let m = obsMsg(dst, o)
                send_id(m); Idle_id(cs, costSend(costNext(ps'), m))

        receiveEvent(m, ps') → TreatMsg_id(m, cs, ps')
```

Extracted from -- section 2. A Generic Modelling Framework in [1]

Figure 2. 4 The Idle Phase

- A physicalStateEvent originated from energy consumption, which is caused by the physical devices, leads to a change of the energy level in physical state.
- A readEnergyEvent produced by processor causes an update of the energy and routing information in computational state, and physical state is changed according to the corresponding cost functions presented in Table 2. 1. If the change of the computational state is significant, then a neighbour message which includes the abstract view of the new computational state is sent out, and both computational state and physical state are updated. Otherwise, this change is ignored and just physical state is updated.
- An observationEvent leads to an execution of the state function next(*cs*) on processor to compute the identifier of the expected destination, and then an observation message can be sent. The physical state is updated with the corresponding cost functions.
- A receiveEvent implies that a message has been received from the network and is waiting for processing in the input-queue, and it is now the front element of the queue. That message will be handled in treat-message phase.

The state changes and message transmission for treat-message phase are given in Figure 2. 5. In this phase the node can deal with an observation message or a neighbour message. After the message is processed a transition to the idle phase is performed, where it can react to further events including the receiving of another message.

```
TreatMsg_id(m, cs, ps) =
        case m of
                obsMsg(dst, o) →
                        if id = dst
                        then let dst' = next(cs)
                                let m' = obsMsg(dst', o)
                                send_id(m'); Idle_id(cs, costSend(costNext(ps),m))
                        else Idle_id(cs, ps)

                neighbourMsg(src, as) →
                        let cs' = updateNeighbourView(cs, src, as)
                        let cs'' = updateRoutingState(cs')
                        let ps' =  costUpdateNeighbourView(costUpdateRoutingState(ps))
                        if transmitChange?(cs, cs'') ∨ ¬consistent?(cs')
                        then let as' = abstractView(cs')
                                let m = neighbourMsg(id, as')
                                send_id(m); Idle_id(cs'', costSend(costAbstractView(ps'), m) )
                        else Idle_id(cs', ps')
```

Extracted from -- section 2. A Generic Modelling Framework in [1]

Figure 2. 5 The Treat Message Phase

- An observation message is processed by checking whether this node is the expected destination for the message. If this is not the case, a direct transition to the idle phase is performed and no change takes place in physical state. Otherwise, the next destination is computed, the observation is forwarded to that destination and the physical state is updated taking the computation and sending costs into account.
- A neighbour message is treated by first updating the neighbour view in the computational state, and then on the basis of this a new routing state is computed. If the change to the computational state is significant, then an abstract view of the computational state should be communicated to the neighbours, and the computational and physical states are updated accordingly. Otherwise, the node returns to the idle phase with a computational state that is just updated with the new neighbour knowledge, and the physical state which is changed by the corresponding cost operations.

This section describes the operational behavior for the normal operations of a sensor node. The energy and neighbour parts appear explicitly through the lifetime of the nodes. Hence it is clear that the generic model reflects energy-aware routing taking the energy level of the nodes into account and using neighbour knowledge, and it is postponed to instantiations of the model to describe how it works.

2.3 Instantiations

In this section two instances of the generic modelling framework presented in the previous section will be introduced. One is DEHAR [8], which is a distributed energy harvesting aware routing algorithm for ad-hoc multi-hop wireless sensor networks, and the other is Directed Diffusion protocol [34], which is a special case of DEHAR.

2.3.1 DEHAR

The DEHAR algorithm is a standard instance of the generic modelling framework, which means it has the detailed description of the state and the operations of the sensor nodes.

The abstract state of a node comprises:

- A *simple distance* $d \in \mathrm{R}_{\geq 0}$ to the base station.
- An *energy-aware adjustment* $a \in \mathrm{R}_{\geq 0}$ of the distance for the route to the base station.

For an abstract state (d, a), we call $\mathrm{dist}(d, a) = d + a$ the *energy-adjusted distance*.

The computational state of a node comprises:

- A *simple distance* $d \in \mathrm{R}_{\geq 0}$ to the base station, like the simple distance of an abstract state.
- An *energy level* $e \in$ Energy.
- An *energy-faithful adjustment* $f \in \mathrm{R}_{\geq 0}$ which is used to keep the node in consistent state.
- A table *nt* containing the abstract state of neighbours.

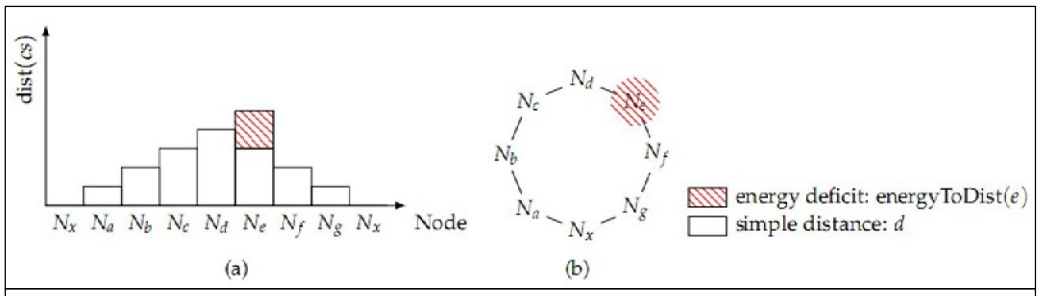
For a computational state (d, e, f, nt), the energy-aware adjustment is partitioned into two parts, where one is an energy adjustment used to convert energy deficiency of the node to a penalized distance by a function *energyToDist(e)* so that less energy means longer distance, while the other is the energy-faithful adjustment used to capture energy deficits along the route to the base station. Therefore the energy-aware adjustment of a computational state is:

$$\mathrm{adjust}(d, e, f, nt) = \mathrm{energyToDist}(e) + f$$

and the energy-adjusted distance of a computational state is:

$$\mathrm{dist}(d, e, f, nt) = d + \mathrm{adjust}(d, e, f, nt) = d + \mathrm{energyToDist}(e) + f$$

The intension of the energy adjustment through the function $\mathrm{energyToDist}(e)$ is to take the available energy level in the node into account. Figure 2. 6 illustrates how the function $\mathrm{energyToDist}(e)$ works on nodes in an example network.

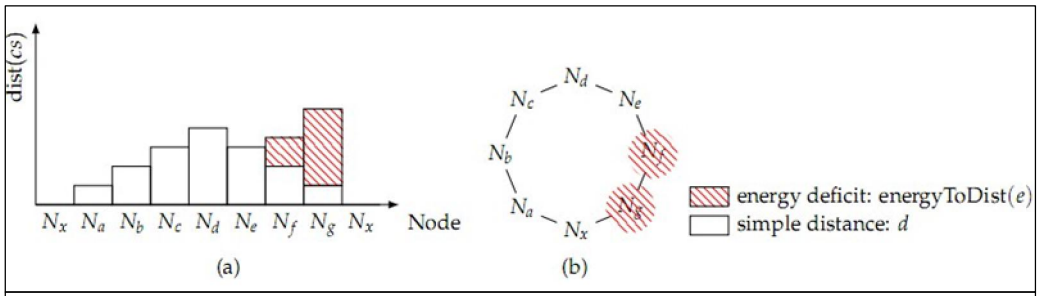


Extracted from -- section 3. Instantiating the Modelling Framework in [1]

Figure 2. 6 The example network with energy adjustment on N_e

In the example network shown in Figure 2. 6 the energy level in node N_e is decreased, then the distance of N_e to the base station is increased by the function energyToDist(e). As a result, the node N_d has only one feasible neighbour N_c left, but all nodes are still in consistent state.

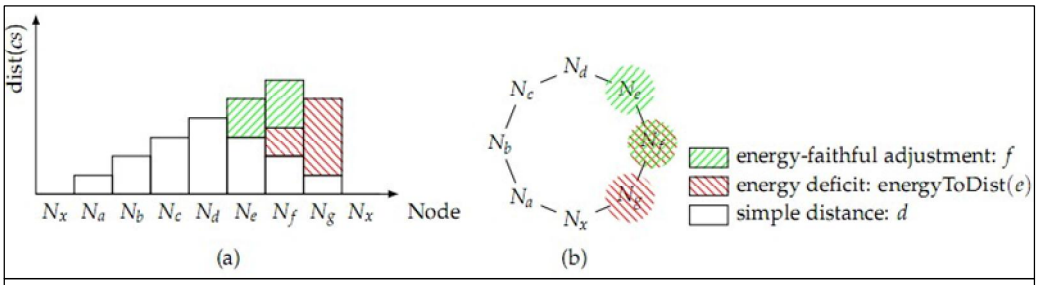
Consider now the situation shown in Figure 2. 7 with energy adjustment on the nodes N_f and N_g. These energy adjustments made by the function energyToDist(e) make the node N_e has inconsistent problem, since its neighbours N_d and N_f both have longer distances to the base station than N_e. It is not difficult to see that if N_e forward observations to its "best" neighbour N_f, N_f would immediately return that observation to N_e since N_e is the "best" neighbour of N_f.



Extracted from -- section 3. Instantiating the Modelling Framework in [1]

Figure 2. 7 The example network with energy adjustment on N_f and N_g

Now the energy-faithful adjustment can be used to deal with this kind of problem. Figure 2. 8 depicts how the energy-faithful adjustment works on inconsistent nodes N_e and N_f. Notice that initially N_f is not inconsistent, but due to the adding of energy-faithful adjustment on N_e, N_f has no feasible neighbour anymore and therefore an energy-faithful adjustment is added on N_f as well.



Extracted from -- section 3. Instantiating the Modelling Framework in [1]

Figure 2. 8 The example network with energy-faithful adjustment on N_e and N_f

The energy-faithful adjustment added on N_e is to make the distance of N_e to the base station becomes K larger than the distance of its "best" neighbour (in this case it is N_f without energy-faithful adjustment) so that N_e keeps consistent. Similarly, the energy-faithful adjustment on N_f makes the distance of N_f becomes K larger than the distance of its "best" neighbour (both N_e and N_g). Now all nodes in the example network reach consistent again. It should be noticed that the value of K need to be selected carefully, as it is used to express the energy deficiencies along a route to the base station.

The DEHAR algorithm also specifies the detailed definition of the physical state, the functions and the operations based on the generic modelling framework, but they are not introduced in this Master thesis since they are not the core of this algorithm.

2.3.2 Directed Diffusion

The routing algorithm Directed Diffusion (DD) [34] is a simple instance of the generic modelling framework since

- The distance of the node is the number of hops to the base station.
- The routing is static, which means that the observations are always routed along the shortest path.

It is obvious that DD algorithm does not support energy-aware routing, but DD provides nodes with an energy harvesting capability. Therefore DD is a good model of reference for comparison with energy harvesting aware routing algorithms like DEHAR. The total energy consumption in a DD network should be smaller than the total energy consumption of DEHAR network, since the shortest path is also the least energy consuming path. However, energy harvesting aware routing algorithm prolongs the lifetime of the network, because the drain of low-energy nodes can be prevented, or at least postponed, and also that nodes get more chance to recover through energy harvesting capability. So, it would be interesting to investigate that how much more energy is required to executing DEHAR algorithm than DD, and how long time the lifetime of the network can be prolonged by using DEHAR algorithm.

These issues have been experimented through UPPAAL simulation and verification of the formal model and will be presented in Chapter 5.

CHAPTER 3.

Extensions to the Generic Framework

This chapter presents some extensions of the generic modelling framework in aspects of communication medium, energy model and environment model, which are indispensable when formalizing the model of energy harvesting aware WSNs.

3.1 Communication Medium

Apart from the model of sensor nodes, the generic framework should have a description of the communication medium through which the nodes store and deliver messages. This medium should at least determine which nodes can receive a message sent by a given node in a given state. It may depend on the available energy, the geographical position, the distance from the senders, and a variety of other parameters [1].

Since in the assumption of the generic framework the sensor nodes are equipped with low-power, low-range radios, the communication medium should be modeled as a radio broadcasting mass medium that are intended to reach a large number of nodes. However, modelling a global medium covering all of nodes in the network is not a convenient way, as a node may only be able to communicate with a small subset of the other nodes, due to the range of radio signal is limited by energy concern. Moreover, a global medium is not enough flexible to reflect the impact of each node's dynamic energy level on its signal strength. Therefore, an individual communication medium for each node seems a good choice and it should be updated according to the energy state of the node.

When the sensor nodes are first used, they will construct their initial communication media,

and that will be only determined by the geographical positions of the nodes if their batteries are fully charged initially. It would be natural to extend the model of sensor nodes with an initialization phase where nodes build up the knowledge of the environment needed to start normal operations through repeated communications. This knowledge should not only reflect which nodes can receive a message sent by a given node when do not take energy into account, but also the physical distances between the node and its neighbours.

During the lifetime of a node, its energy level is dynamically changed through the occurrence of the associated operations. If the transmitted power of the transmitter is adjusted automatically according to the residual energy, the communication medium of the node needs to be updated when the energy change is significant. Thus, this update can be incorporated in readEnergyEvent which is introduced in Subsection 2.2.3. Considering a particular neighbour, which is reachable for the node initially but too far for the attenuated radio signal, it should be excluded from the medium until the signal strength recovers.

The communication model should capture the initial state of the medium which is built in initialization phase, and the updates of the medium caused by dynamic energy changes. A detailed description of the communication model is given as follows:

- The communication medium could be included in a node's computational state as a NeighbourSet, in which the elements have the form of (*id, pdst*), where *id* is the identifier of a neighbour and *pdst* is the physical distance between the neighbour and the node.
- In initialization phase, a node builds up its NeighbourSet by broadcasting repeated query messages. Each node that receives this query will reply its identifier and geographical information. There is an assumption that all nodes have same radio signal strength initially, so that the reply can reach the sender of the query. After the node receives the reply, the physical distance between them is computed and recorded in NeighbourSet with the corresponding identifier.
- A node needs to update its NeighbourSet when a readEnergyEvent is originated and the predicate transmitChange? is true. This update is based on the physical distances to its neighbours and the updated radio signal range caused by dynamic energy changes.
- A node should not delete the record of a neighbour from its NeighbourSet even if that neighbour is no longer considered as one, as the energy level of the node may be restored through energy harvesting. Therefore, a flag could be used to indicate that a neighbour is temporarily discarded.

Figure 3. 1 illustrates the change of the communication medium for a particular node N_1 in a sample network. Initially, the NeighbourSet of N_1 is {(2, 100), (3, 120), (4, 150), (5, 150), (6, 110)} because the signal range is 160. When the signal range of N_1 is attenuated to 130, N_4 and N_5 are no longer considered as neighbours of N_1 since their physical distances to N_1 are larger than 130, so the NeighbourSet changes to {(2, 100), (3, 120), (-4, 150), (-5, 150), (6, 110)}. The negative sign added to the identifier of N_4 and N_5 is a flag to indicate these two nodes are out of the current signal range of N_1.

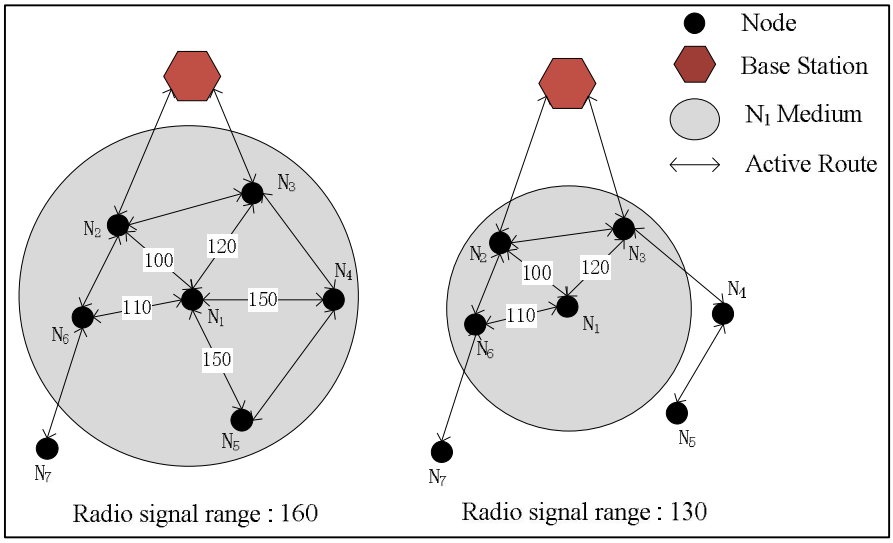


Figure 3. 1 the illustration of communication medium change

Since there are several transceivers connected to a communication medium in practice, the communication model should capture a collision protection mechanism which is used to resolve potential contention and collision among nodes on using the medium. Such a mechanism is usually provided in Medium Access Control (MAC) data communication protocol. In this Master project it is assumed that there is no collision happens by giving an individual medium to each node, since we consider that the low-level protocol of WSNs will take care of this problem. More introduction of this kind of protocol for low-level communication can be found in Chapter 6.

3.2 Energy Model

For each node in the network, the energy consumed and produced is recorded by an energy model in physical state. There are two parts of the energy model, where one is a battery used to supply energy for physical devices like a sensor or a receiver, while the other is a capacitor used to store energy produced by energy harvesting. In the energy model the battery is fully charged initially and can be recharged by the capacitor during the operation of the node. The rationale for having two energy containers rather than a combined one, e.g. a super capacitor, is that the consumption of the battery is predictable and is not a very frequent event, whereas the harvesting of energy is a continuous and unpredictable process as it depends on volatile environment conditions. Therefore, modelling them separately helps to have a clear knowledge of the energy state in a node. In addition, it is easier to construct energy model in this way in UPPAAL comparing to modelling these two energy containers in combination.

The overview of the energy model is shown in Figure 3. 2. The left box symbolizes an energy

harvester, e.g. a solar panel, which is connected to a capacitor so that the energy collected from the environment can be stored. In the middle the capacitor and a rechargeable battery constitute the energy model together. There is a connection between them so that energy stored in the capacitor can be transferred to the battery. To the right the devices which consists of a processor, a sensor, a transmitter and a receiver, are consuming the energy from the battery in order to maintain the normal operations of a node.

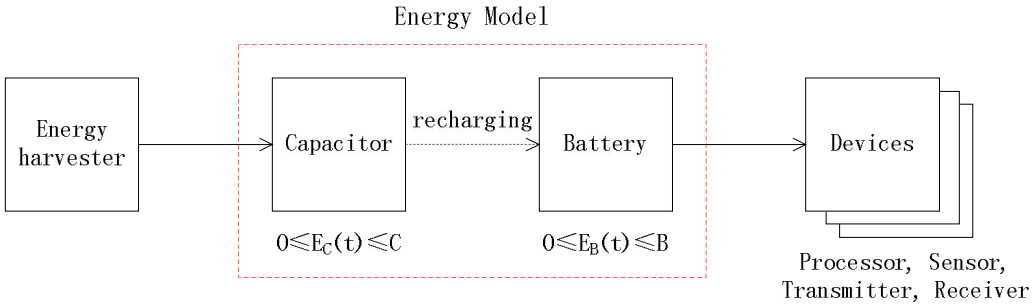


Figure 3. 2 Overview of the energy model

There is an upper bound of the capacity of each energy container in energy model, i.e.

$$0 \leq E_C(t) \leq C \quad (1)$$
$$0 \leq E_B(t) \leq B \quad (2)$$

where C is the capacity of the capacitor and $E_C(t)$ is the energy stored in capacitor at time t. In a similar way, B is the capacity of the battery and $E_B(t)$ is the energy stored in battery at time t.

In the following, the details of the capacitor and the battery will be introduced separately.

3.2.1 Capacitor

There are two tasks for the capacitor in energy model. One is storing the energy collected from the environment by energy harvester. Ideally the capacitor itself has a perfect performance, thus the rate of energy charging in capacitor only depends on the work of energy harvesting device. For the capacitor the energy charging model is

$$E_H(t_1, t_2) = \int_{t_1}^{t_2} P_H(t)\, \delta t \quad (3)$$

for $t_1 < t_2$, where $P_H(t)$ is the power harvested by the energy harvester and $E_H(t_1, t_2)$ is the energy harvested in time interval from t_1 to t_2. The power harvested $P_H(t)$ is determined by the efficiency of the energy harvesting device and the environment conditions.

The other task of the capacitor is to transfer energy to the battery. This transfer may have a constant rate and obviously it stops when the capacitor is empty or the battery is fully charged. So, the energy reduction of the capacitor is modeled as

$$E_T(t_1,\ t_2) = \mu(t_2 - t_1) \tag{4}$$

where μ is the constant rate and $E_T(t_1, t_2)$ is the energy transferred from the capacitor to the battery during time interval (t_1, t_2). Since transferring energy to the battery is the only energy consumption of the capacitor, the energy stored in capacitor can be modeled as

$$E_C(t_2) = E_C(t_1) + E_H(t_1,\ t_2) - E_T(t_1,\ t_2) \tag{5}$$

Notice that in equation (5), $E_H(t_1, t_2)$ should be set to zero if the energy stored in capacitor arrives the upper bound of the capacity, and $E_T(t_1, t_2)$ should be set to zero if the capacitor is empty or the battery is full.

3.2.2 Battery

To construct the battery model, detailed information about the cost functions listed in Table 2. 1 is needed. The process of energy consumption of the battery relies on the duration and power consumption of the eight cost functions. All these parameters are customizable. The energy consumption model of the battery is

$$E_D(t_1,\ t_2) = \int_{t_1}^{t_2} P_D(t)\ \delta t \tag{6}$$

for $t_1 < t_2$, where $P_D(t)$ is the power consumed by the battery and $E_D(t_1, t_2)$ is the energy consumed in time interval from t_1 to t_2. At a certain time instant, $P_D(t)$ is the sum of the power consumed by the activated devices. Figure 3. 3 depicts an example of energy consumption of the battery when only sensor, transmitter and receiver are involved in a node's normal operation.

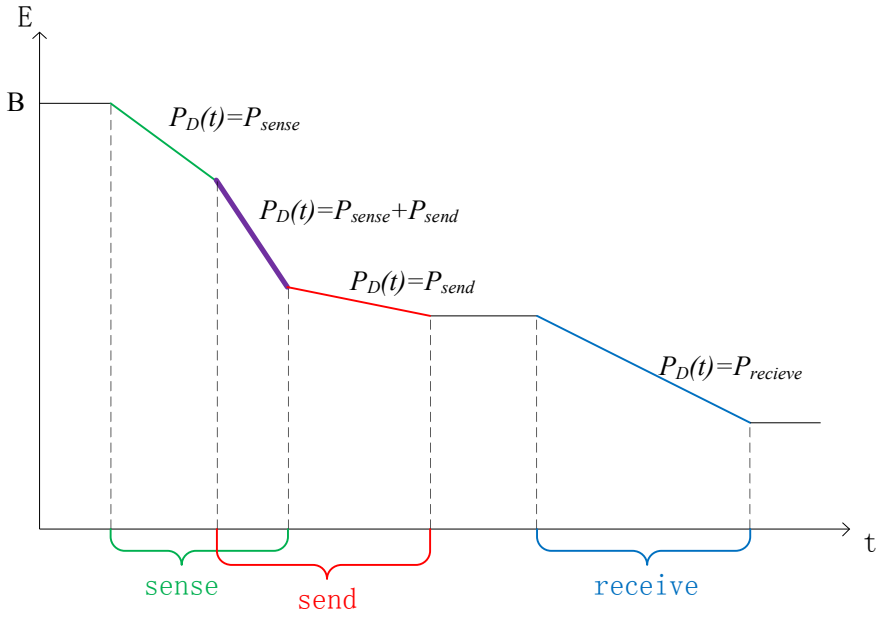


Figure 3. 3 An example of energy consumption in battery model

It is clear that initially the battery is fully charged and its capacity is B. Soon later the sensor starts working, so the energy in the battery keeps reducing at a rate of the sensor's consuming power until the transmitter is activated as well. When both the sensor and the transmitter are in active state, $P_D(t)$ becomes the sum of the power consumed by them. As soon as the sensor finishes its job, $P_D(t)$ changes to the power of the transmitter solely. If there is no devices consuming energy any more, $P_D(t)$ will be set to zero, like in the time interval between the time point at where the transmitter stops working and the time point at where the receiver starts working.

On the other hand, since the energy transferred from the capacitor is the only source of the energy charged to the battery, the battery can be modeled as

$$E_B(t_2) = E_B(t_1) + E_T(t_1, t_2) - E_D(t_1, t_2) \tag{7}$$

Notice that in equation (7), $E_T(t_1, t_2)$ should be set to zero if the battery is fully charged or the capacitor is empty. The node will die or at least go to sleep state if $E_B(t_2)$ is less than or equals to zero.

3.3 Environment Model

The environment model describes the renewable environmental resource which is used by energy harvesting device. Generally the environment model can be partitioned into two parts,

a uniform energy source and a non-uniform attenuation [8]. The uniform energy source $P_I(t)$ is the power production by energy harvesting device under ideal conditions. Here "ideal" means that for a particular kind of environmental resource, the energy harvesting device operates under standard related environment conditions without interference. For instance, considering the environmental resource is solar, $P_I(t)$ will be the power production by a solar panel under a clear sky at midday. The non-uniform attenuation $f_I(t)$ is the obstacles in energy harvesting process. These obstacles come from variable environmental factors and human interference. Such obstacles could be variable sunlight intensity, local weather conditions, shadows from buildings and trees, etc. when considering solar energy harvesting.

The power production $P_H(t)$ (which is also referred in equation (3)) of the energy source is modeled as

$$P_H(t) = P_I(t) f_I(t) \tag{8}$$

There is an assumption that the efficiency of the energy harvesting device is excellent and is not affected by the environment conditions, so it is not involved in equation (8). The model described above is a generic framework of the environment model, and in the following a concrete example will be introduced.

The environmental factors which can affect the efficiency of energy harvesting process are completely different when distinct types of environmental resources are utilized, like if the energy source is from vibration instead of insolation, the factors will change to wind conditions and building location. It is therefore difficult to establish a detailed environment model that covers a broad range of environmental resources. In this Master thesis, an environment model used for solar energy harvesting is selected to be built.

The solar environment model describes the insolation on each node's energy harvester, i.e. a solar panel. The uniform energy source $P_I(t)$ is generated from the value of insolation on the solar panel under clear sky at midday. The model should also capture that the related environmental factors, which are local weather conditions and shadows from buildings and trees, lead to attenuation $f_I(t)$ on the power production. In order to do so, the insolation on each node can be simply divided into three scenarios as follows:

- Under clear sky: the power production $P_H(t)$ is equal to $P_I(t)$. It is assumed that the difference of sunlight intensity caused by time is ignored, and only daytime and nighttime are distinguished in the model. During nighttime, $P_H(t)$ is set to zero.
- Under light shadow: the light shadow is caused by clouds, trees and buildings. Usually this kind of shadow appears irregular and has a short lasting time. Since there is still sunlight diffusing in the air, the attenuation originated by light shadow is slight. So, it is reasonable to assume the non-uniform attenuation $f_I(t)$ is 0.6, then the power production $P_H(t)$ is equal to $0.6P_I(t)$.
- Under Strong shadow: the strong shadow is caused by rainy (or snowy) day. Compared to light shadow, this kind of shadow lasts longer and brings a larger attenuation. In this

scenario $f_I(t)$ can be set to 0.3, and correspondingly the power production $P_H(t)$ is equal to $0.3P_I(t)$.

To capture the occurrence of these three scenarios, the statistics of insolation and weather conditions for a particular geographical position, which is in terms of average clear sunshine hours, average light shadow hours and average strong shadow hours, is needed. The statistics of sunshine hours for Copenhagen during a year is shown in Figure 3. 4.

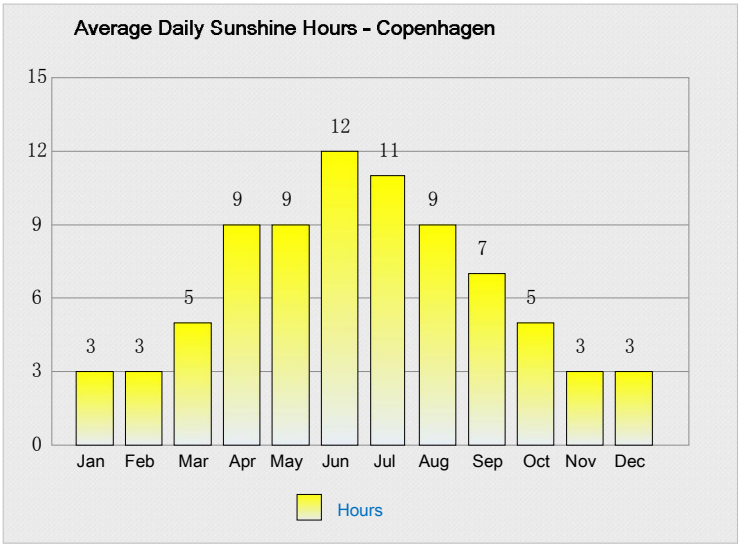


Figure 3. 4 Average daily sunshine hours in Copenhagen

The average daily sunshine hours implies the insolation time of a node deployed in Copenhagen under clear sky and light shadow in a particular month. Due to the irregular and short-lasting characteristics of light shadows, the average light shadow hours can be generated from a random value which is from 1/4 to 1/3 of the average daily sunshine hours shown in the figure, and the rest is the insolation time of the node under clear sky. For instance, in January, the average daily sunshine hour is 3. If the light shadow hour is 1/3 of the sunshine hours, the average clear sunshine hours will be 2 and the average light shadow hours will be 1.

On the other hand, the average strong shadow hours can be calculated from Figure 3. 4 and Figure 3. 5.

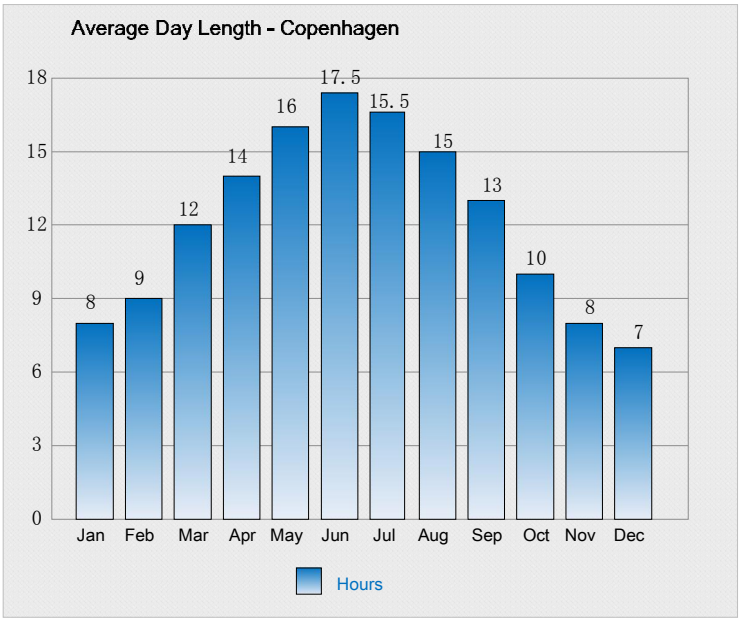


Figure 3. 5 Average day length of Copenhagen

Figure 3. 4 and Figure 3. 5 give the average daily sunshine hours and the average day length in Copenhagen respectively. It is not difficult to imagine that the difference between them is the average strong shadow hours caused by rain and snow. For example, in January, the average day length is 8 and the average daily sunshine hour is 3. Therefore, the average strong shadow hours can be estimated to 5. The solar environment model for a node deployed in Copenhagen in January is summarized in Table 3. 1:

Time	**Environmental Factors**	**Duration (hours)**	**Power Production $P_H(t)$**
day	clear sky	2	$P_I(t)$
	light shadow	1	$0.6P_I(t)$
	strong shadow	5	$0.3P_I(t)$
night	-	16	0

Table 3. 1 Solar environment model for nodes in Copenhagen in January

Notice that the duration of each of the scenarios listed in above table is accumulative time, as the weather changes continuously. Figure 3. 6 gives an example of the energy production in capacitor based on this solar environment model. The continuous change of the environmental factors should be reflected in the environment model, to express the reason of the dynamic energy change of the node's energy harvester.

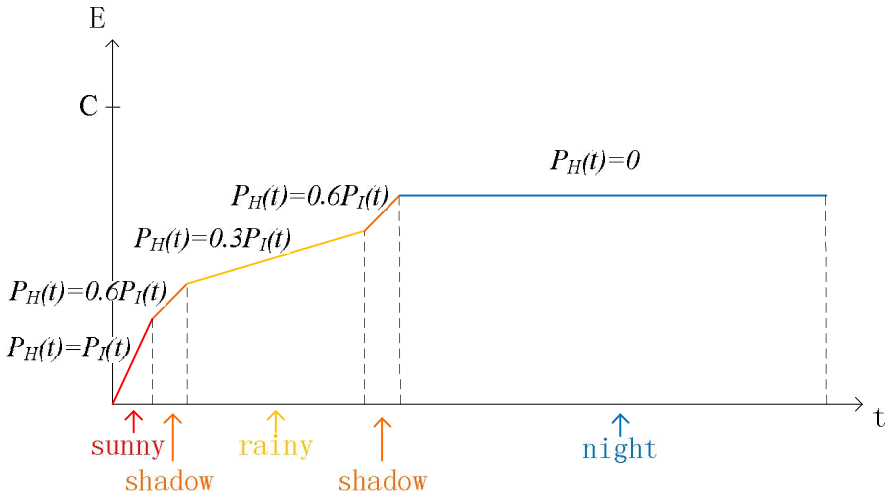


Figure 3. 6 An example of energy production in capacitor model

The solar environment model is based on the statistics of the weather and day length of a particular geographical position. These statistical data are collected from professional meteorological agencies. The model should generate the value of $P_I(t)$ and $f_I(t)$ through real-life experiments, and it is assumed in this Master thesis for simplicity.

3.4 Instantiation

The purpose of this section is to introduce a simple instantiation, which is called Simple DEHAR. As an intermediate instance between DEHAR and DD, it is produced during the Master project to help conduct concrete experiments from scratch.

Simple DEHAR

The simple DEHAR algorithm is a condensed version of DEHAR algorithm and it supports energy harvesting and incomplete energy-aware routing. It is introduced since the energy-faithful adjustment originated by inconsistent nodes in DEHAR algorithm leads to a large number of neighbour message communications. It is difficult to guarantee that the gain by using it can exceed the cost of these communications at the first stage. Therefore, a simple version of DEHAR algorithm, in which no inconsistent node happens, is produced during the Master project.

In order to avoid inconsistencies, the simple DEHAR algorithm has an assumption that the energy adjustment made by the function energyToDist(e) is smaller than the radio signal range. It means the energy-aware adjustment due to energy deficiency only has impact on which neighbour is the best node to forward observations, but will not lead to a situation that

there is no best neighbour exists. As shown in Figure 3. 7, the example network has a layer structure, each node can communicate with the nodes in its layer and its adjacent layer. It is illustrated that the nodes N_c and N_d change their best neighbours from N_a to N_b due to the energy dificiency on N_a, but this kind of energy-adjusted distance will never bring inconsistent nodes.

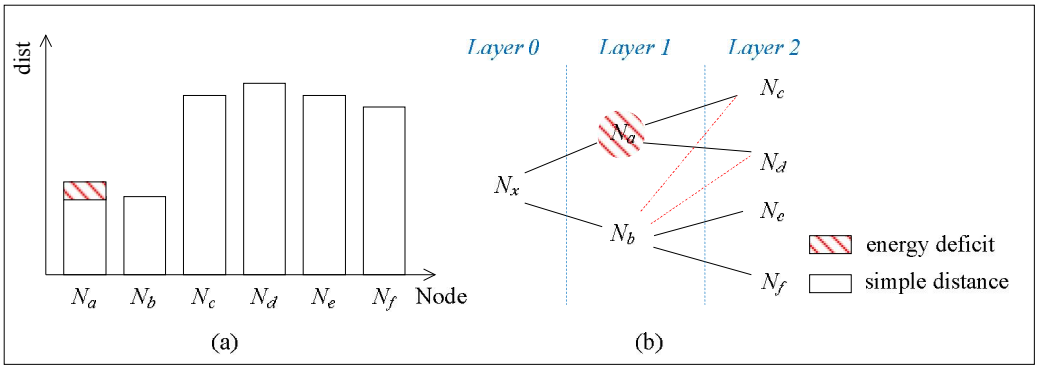


Figure 3. 7 The example layer network with energy adjustment on N_a

The simple DEHAR algorithm support incomplete energy-aware routing, here “incomplete” means that the simple DEHAR can only change a route for transferring observations in the same direction to keep low-energy nodes avoided, and it will have no effect if all routes in one direction are in low-energy state. Actually it is the energy-faithful adjustments on inconsistent nodes in DEHAR algorithm responsible for changing the direction of forwarding observations.

CHAPTER 4.

UPPAAL Model of WSN Framework

In this chapter, a formal UPPAAL model of energy harvesting aware WSNs based on the generic modelling framework presented in previous chapters is provided. First, the structure of the WSN model is explained in the form of semantics. This semantics helps to have a clear understanding of the WSN model. Then the detailed explanation for each component of the model is given. Representing the WSN framework in UPPAAL enables tool-support for simulation and verification through model checking, and some experiments will be introduced in Chapter 5. It is assumed that the reader has the general knowledge of timed automata and UPPAAL.

4.1 Structure of the WSN model

The WSN model is expressed as a collection of timed automata, combined in parallel and communicating with each other through shared variables and synchronizations.

The structure of the WSN model is described as follows:

$$WSN = \|_{i=1}^{n} Node_i$$
$$Node_i = Components_i \| Energy_i \| Environment_i \| Medium_i$$
$$Components_i = Sensor_i \| Transmitter_i \| Receiver_i \| Harvester_i \| Processor_i$$

where $\|$ denotes parallel composition of the timed automata.

This structure of the WSN model follows the description of the generic framework. The

UPPAAL network is consisted of a number of sensor nodes ($Node_i$) which are modeled based on the conceptual fundament introduced in Section 2.2. As mentioned, each node has a sensor, a transmitter, a receiver, a processor and an energy harvester as the physical components ($Components_i$). In UPPAAL, a sensor node comprises one model for each of these components. These models will be presented in Section 4.2. It is not difficult to see that the UPPAAL network can be achieved by giving identity to such node.

Apart from the physical components, each node has description of its individual communication medium ($Medium_i$) in UPPAAL. A medium determines which nodes can receive a message sent by the node, but the problems of message loss and collision due to medium unreliability are not considered (for simplicity) in the UPPAAL model. This description of the medium will be given in Section 4.3.

To complete the formalization of the WSN model, an energy model ($Energy_i$) and an environment model ($Environment_i$) for each node are necessary. The reason to have an individual environment model for each node is that the weather conditions for spatially distributed nodes may be different. Section 4.4 and Section 4.5 will provide these two models respectively.

4.2 Components

The physical components of a node can be explained in terms of the following parallel composition of timed automata.

$$Components_i = Sensor_i \| Transmitter_i \| Receiver_i \| Harvester_i \| Processor_i$$

All of these timed automata except for the harvester can be represented in classical UPPAAL. Since the harvester is connected to the environment model, some special features of SMC extension of UPPAAL are needed. The following subsections explain these four classical timed automata of a node with identity (***id* = 1**) in detail. The timed automaton for the harvester will be presented together with the environment model in Section 4.4.

4.2.1 Sensor

This component is created to provide the model of the physical sensor part of a node. It keeps the same assumption that the sensor is not biased towards any particular kind of observation, which is also mentioned in the generic framework. In short, the timed automaton contains three locations and three transitions. The timed automata for the sensor ($Sensor_1$) can be seen in Figure 4. 1.

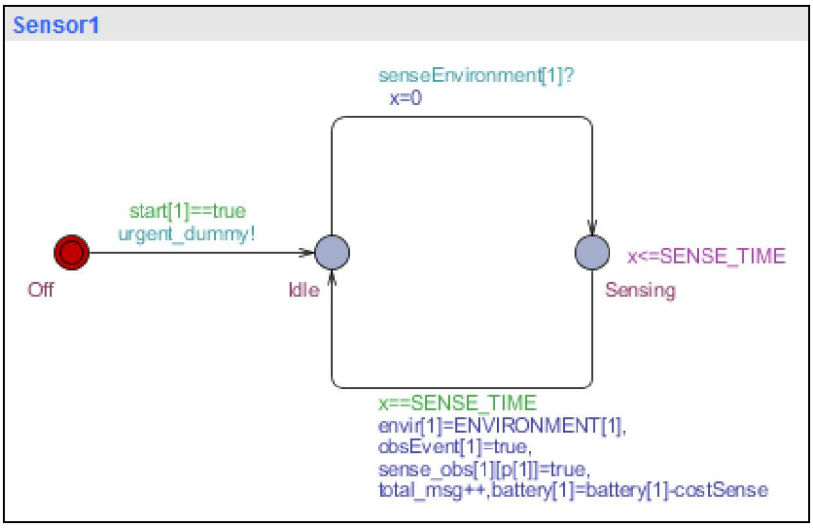


Figure 4. 1 UPPAAL model for the sensor

More detailed comments to each of the locations and the transitions are given below:

- **Off** - which models the physical off state of the sensor.
- **Idle** - which models the inactive state of the sensor.
- **Sensing** - which models the sensing process of the sensor. The sensor should stay in this location for **SENSE_TIME** to complete its collection task.

Initially the node is closed so the sensor stays in **Off** location. As soon as the node is switched on, the sensor goes to **Idle** location and waits for collection instruction from the processor. The transmission of this instruction is achieved by synchronization between the processor and the sensor via channel **senseEnvironment(*id*)**. If receiving this instruction, the sensor will go to **Sensing** and start sampling the environment. When it finishes the task, the sensor records the result and notifies the processor that there is a pending observation by an **observationEvent(*id*)** introduced in Figure 2. 2, and also originates the cost function **costSense** presented in Table 2. 1. After that, the sensor makes a transition to **Idle** location and waits for another collection instruction.

4.2.2 Transmitter

This component is created to provide a concrete procedure of how a message is sent out. There are two kinds of messages that need to be transmitted, one is observation message, and the other is neighbour message. As explained in Subsection 2.2.4, for different kinds of messages, different information is required to support their transmission, like the destination of observation message and the source of neighbour message. Therefore, the UPPAAL model of the transmitter has two parts for these messages. The timed automaton for the transmitter

(Transmitter$_1$) is shown in Figure 4. 2.

As mentioned in Subsection 2.2.4, each node maintains an output-queue for storing the messages that need to be sent out. If a new message is added to the queue, a flag **p[*id*]** will be increased 1 to record the new pending message. The transmitter can broadcast the message which is in front of the queue, here the front element is traced by another flag **q[*id*]**.

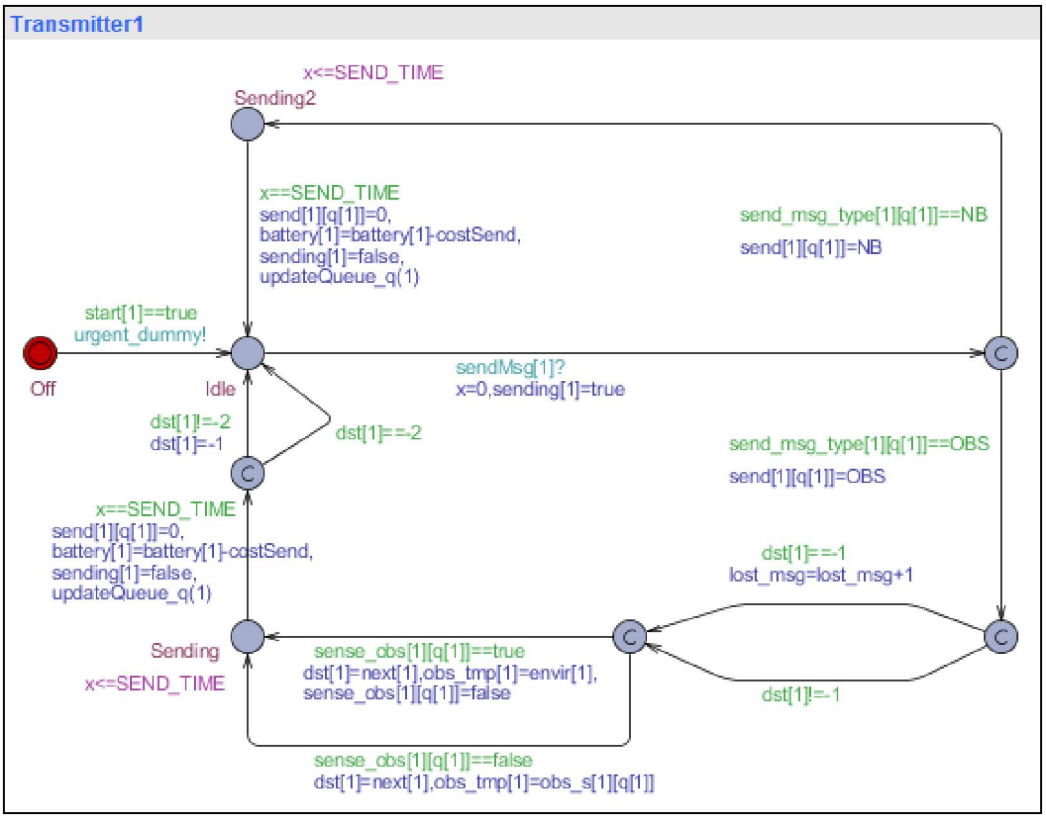


Figure 4. 2 UPPAAL model for the transmitter

The locations and the transitions are explained as follows:

- **Off** - which models the physical off state of the transmitter.
- **Idle** - which models the inactive state of the transmitter.
- **Sending** - which models the sending process of the observation message. The transmitter should stay in this state for **SEND_TIME** to complete its transmission task.
- **Sending2** - which models the sending process of the neighbour message. The transmitter should stay in this state for **SEND_TIME** to complete its transmission task.

As was the case with sensor, the transmitter goes to location **Idle** from location **Off** when the node is turned on. A synchronization channel **sendMsg(*id*)** is used to deliver the sending order from the processor to the transmitter. If the transmitter receives an order for sending out an observation message, the destination of the observation will be computed and broadcasted together with the content of the observation. If the order is sending out a neighbour message,

the transmitter will broadcast the node's identity and abstract state.

Two circumstances of the observation message to be sent are shown on the transitions connected to location **Sending**, where one is concerned with the observation sampled by the sensor, while the other is concerned with resending observations from other nodes in the network.

After broadcasting the message for **SEND_TIME**, the front element of the output-queue is updated and the related cost function **costSend** is executed. Finally, the transmitter makes a transition to **Idle** location and waits for another sending order.

Notice that the assignment of the variable **dst[*id*]** to -1 and -2 on transitions is just to gather statistics of the lost messages. It has no effect on the work of the transmitter.

4.2.3 Receiver

When a message is sent to the network by a node, the receivers of other nodes, which are deployed within the radio signal range of the sender, may obtain the message. Notice that there may be more than one receiver got the message due to the broadcast communications. Figure 4. 3 contains the timed automaton for the receiver ($Receiver_1$).

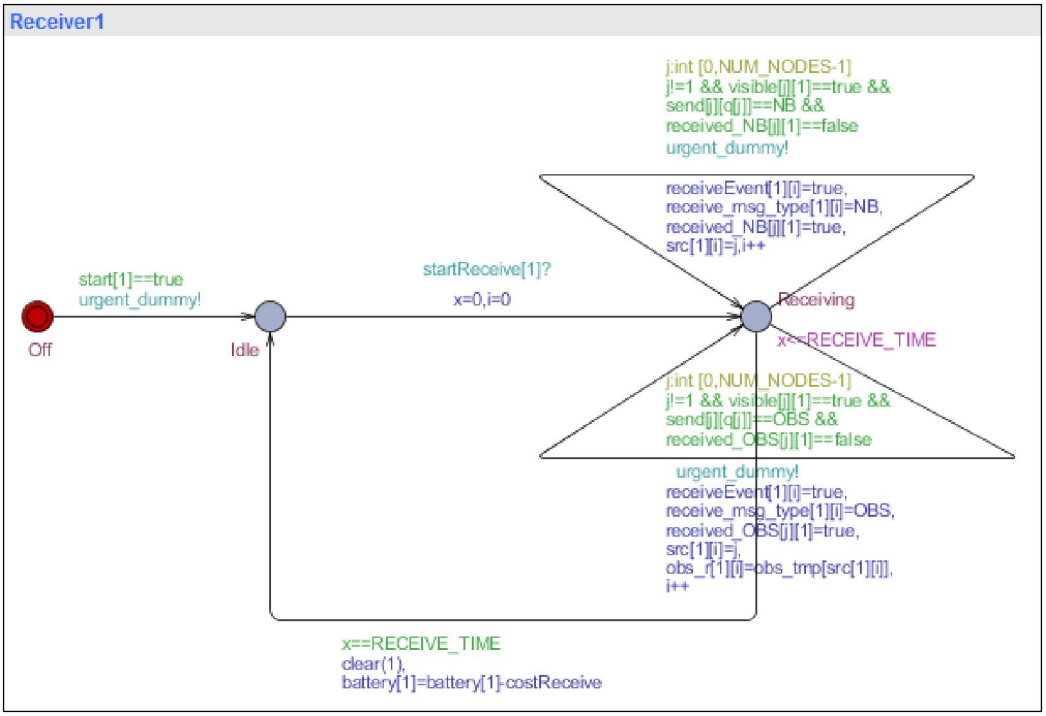


Figure 4. 3 UPPAAL model for the receiver

As was the case with sensor and transmitter, the locations **Off** and **Idle** provide the administration regarding the start of the receiver. In **Idle** location the receiver is basically inactive waiting for some receiving commands from the processor. Once a command is delivered through synchronization channel **startRecieve[*id*]**, the receiver will make a transition to **Receiving** location, at which two kinds of messages can be accepted, and stay there for **RECEIVE_TIME**. If the received message is an observation message, the environment data and destination information will be recorded. Otherwise it is a neighbour message, the abstract state and identity of the source node will be recorded.

It is not difficult to see that during the **RECEIVE_TIME**, the receiver may obtain several messages since there may be many neighbours of the node sending out messages during that time. Therefore, an input-queue is needed to store these received messages as introduced in Subsection 2.2.4. In the timed automaton, the second dimension with index ***i*** of those two-dimension arrays is used as the input-queue to store the related data. For example, the second dimension of array **obs_r[1][*i*]** is used to store the observation of the ***i***th received message by **Receiver1**.

Notice that a message is communicated successfully between two nodes only when the receiver is in **Receiving** location while the transmitter is in **Sending** or **Sending2** location. It means that a message can be communicated if and only if there is an overlap between the sending interval and the receiving interval. For simplicity it is assumed that messages are broadcasted and received instantaneously in this UPPAAL model.

After the receiving interval, the cost function **costReceive** presented in Table 2. 1 is executed and a transition to **Idle** location is performed, where the receiver can react to another receiving command.

4.2.4 Processor

This component is created to provide controllers to the sensor, the transmitter and the receiver, and a computational unit to react to events and to implement the energy-aware routing algorithm. The processor can be explained in terms of the following parallel composition of timed automata.

$$Processor_i = P\_Sensor_i \| P\_Receiver_i \| P\_EnergyRead_i \| P\_Unit_i$$

In the UPPAAL model, the sensor and the receiver are designed to work periodically. The **SENSE_TIME** and **RECEIVE_TIME** in the timed automata for these two actuators only decide the active duration of the devices, it is therefore necessary to have controllers to determine the working periods of them.

The timed automaton for the controller of the sensor (P_Sensor_1) is given in Figure 4. 4.

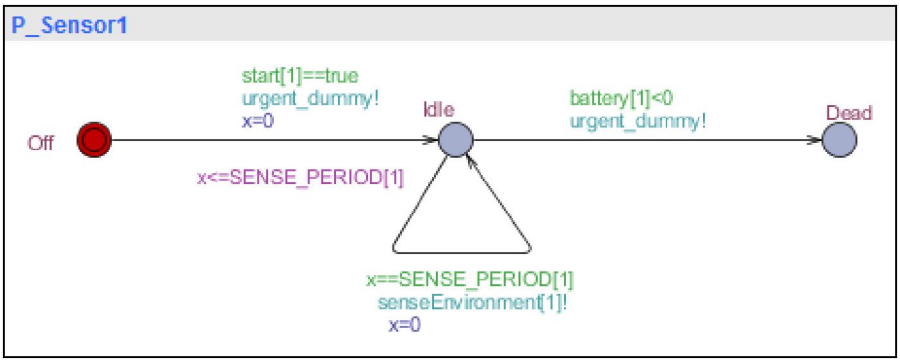


Figure 4. 4 UPPAAL model for the controller of the sensor

After the node switches on, the controller of the sensor stays in **Idle** location. Every **SENSE_PERIOD[*id*]** time the controller sends a collection instruction to the sensor via channel **senseEnvironment(*id*)**. By changing the value of **SENSE_PERIOD[*id*]**, the duty-cycle of the sensor can be adjusted like the example shown in Figure 4. 5.

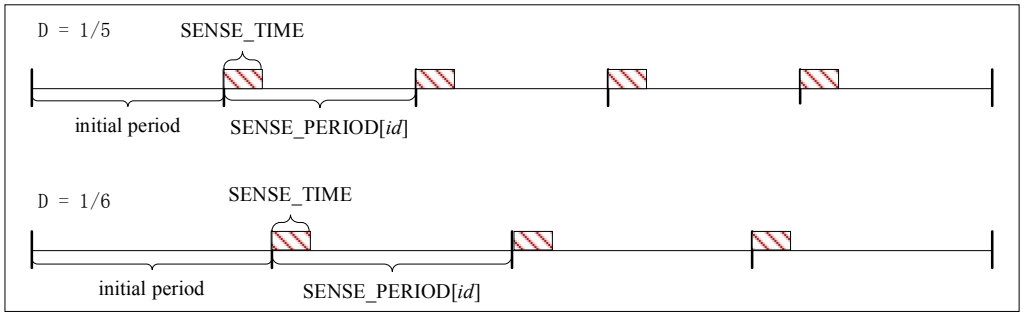


Figure 4. 5 An example of changing the duty-cycle of the sensor

The timed automaton for the controller of the receiver ($P\_Receiver_1$) can be seen in Figure 4. 6.

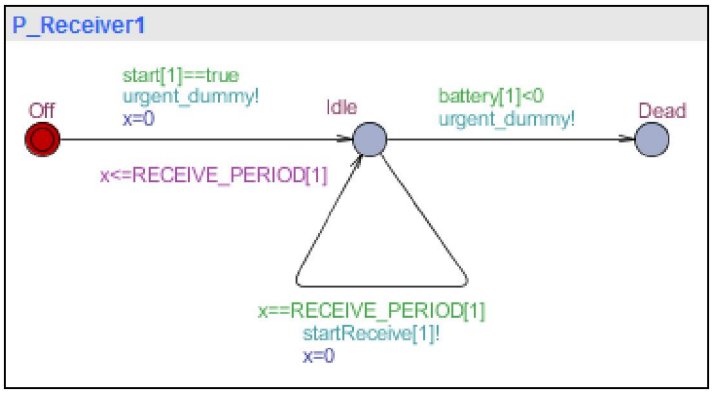


Figure 4. 6 UPPAAL model for the controller of the receiver

Similar to the sensor, the period of the receiver is determined by the variable **RECEIVE_PERIOD[*id*]** in its controller. So the duty-cycle of the receiver can be adjusted as well.

Another task of the processor is to originate the event readEnergyEvent which is used to update the energy state of the node. The timed automaton for this part ($P\_EnergyRead_1$) is shown in Figure 4. 7.

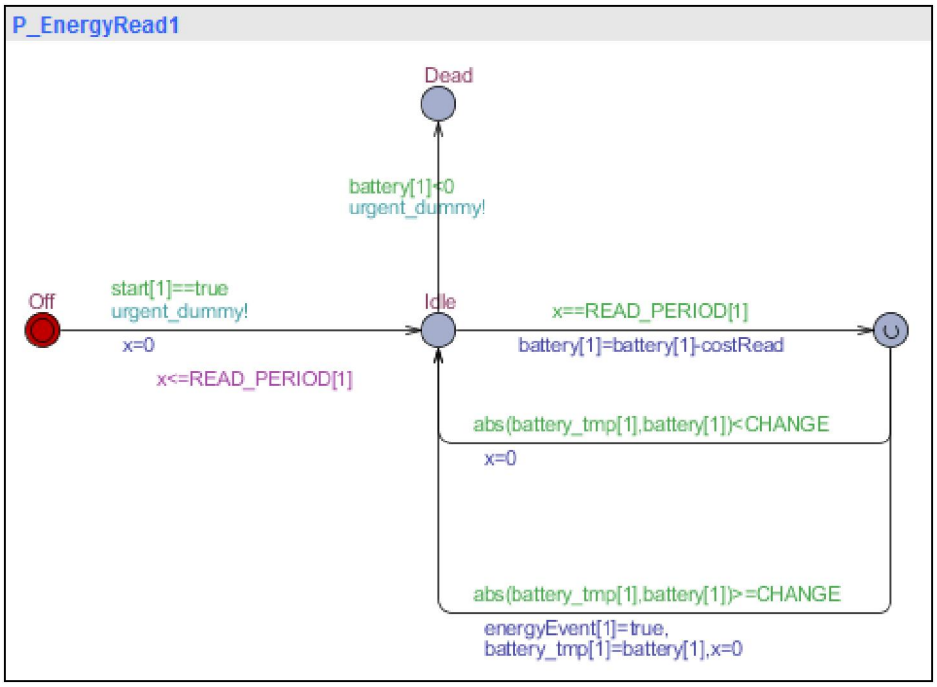


Figure 4. 7 UPPAAL model for the initiator of readEnergyEvent

It is a delicate matter to decide when to originate a readEnergyEvent because a risk is that too much reading costs a lot of energy, and hence too much energy is spent on the administrative issue. In this UPPAAL model, readEnergyEvent is designed as a periodical event and the appropriate period **READ_PERIOD[*id*]** is obtained from experiments.

As presented in the generic framework in Figure 2. 4, when a readEnergyEvent is originated, the energy part of the node is updated. If the change is insignificant then this change is ignored. Otherwise, the abstract view of the new energy state needs to be sent to the neighbours. Here the variable **energyEvent[*id*]** is used to notify the computational unit of the processor that a neighbour message is needed to be produced to send the abstract view out.

It should be noted that there is no customized controller for the transmitter, as the transmitter will be activated as long as there are messages waiting in the output-queue. The delivery of the instruction to activate the transmitter is involved in the computational unit. Figure 4. 8 gives the timed automaton for the computational unit of the processor (P_Unit_1) and Figure 4. 9 gives the part of the activation of the transmitter involved in computational unit.

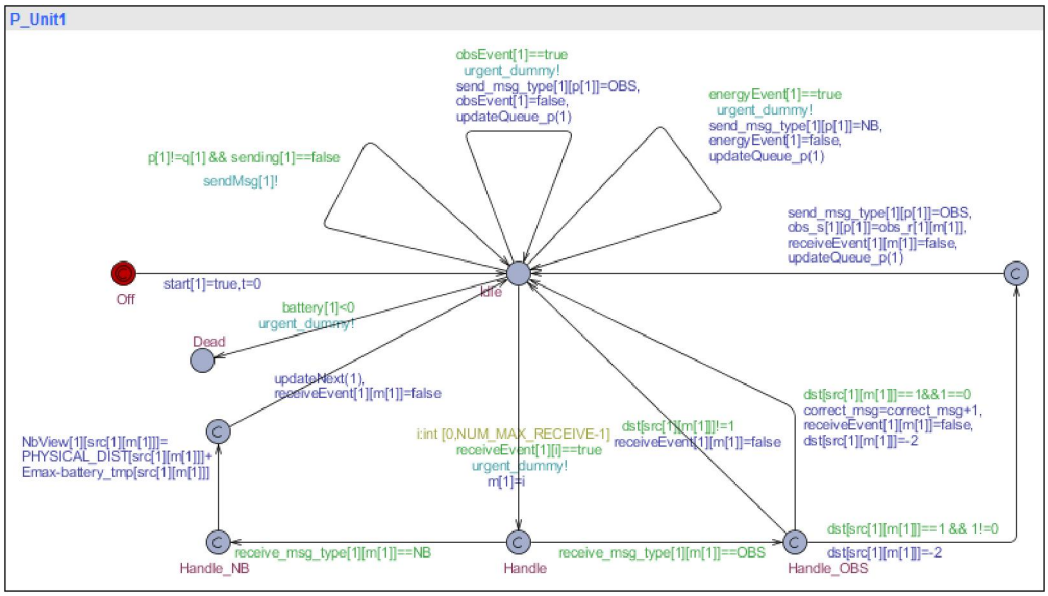


Figure 4. 8 UPPAAL model for the computational unit

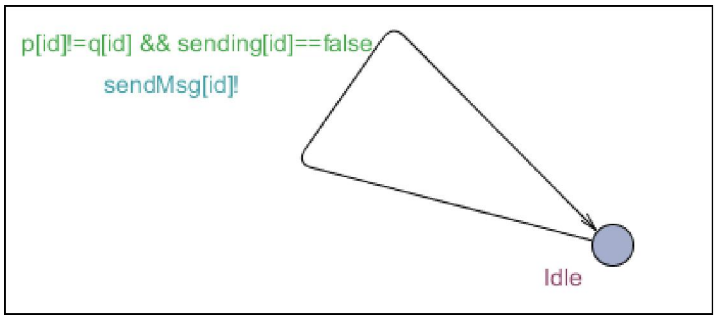


Figure 4. 9 The activation of the transmitter

In order to provide a better understanding, the computational unit can be divided into three parts, one for reacting to each of the events from Figure 2. 2 except physicalStateEvent – observationEvent, receiveEvent and readEnergyEvent. Since physicalStateEvent is just about the energy change caused by the actuators and the sensor, it is modeled as the cost functions instead of an event occurring in the processor for simplicity.

Reacting to observationEvent

Figure 4. 10 shows the part of the treatment of observationEvent in the UPPAAL model of the computational unit. An observationEvent is originated by the sensor of this node due to a new observation is sampled.

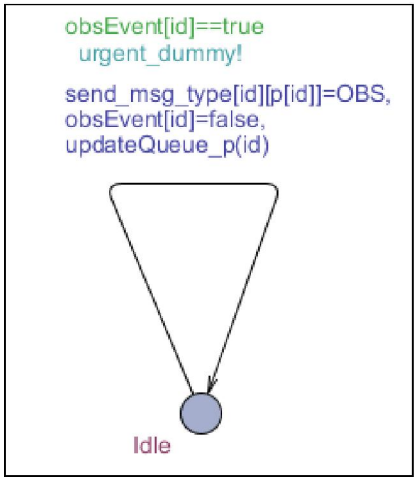


Figure 4. 10 The treatment of observationEvent in computational unit

Corresponding to the treatment of observationEvent in the generic framework shown in Figure 2. 4, a new observation message is generated and put into the output-queue waiting to be sent out. The only difference between this UPPAAL model and the original framework is that the destination of this observation message is not computed during the treatment of the event, but computed when this message is ready to be sent out. This improvement was made because the best neighbour of this node may be changed while the message is waiting in the output-queue due to the dynamic energy-aware routing algorithm.

Reacting to receiveEvent

Figure 4. 11 gives the part of the treatment of receiveEvent, which is related to the treat-message phase in the generic framework given in Figure 2. 5. From the automaton we can see that as long as there are messages waiting in the input-queue, a receiveEvent will be offered by the computational unit.

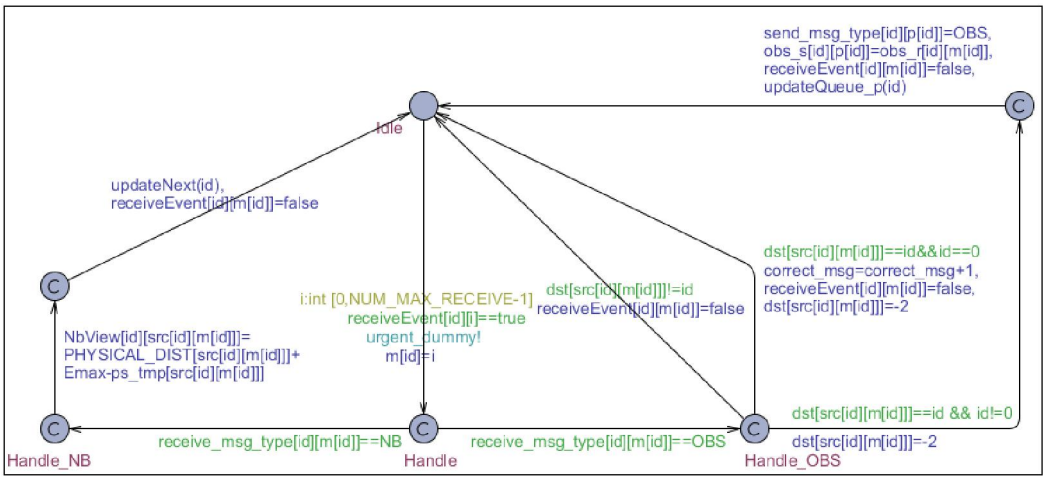


Figure 4. 11 The treatment of receiveEvent in computational unit

In this timed automaton the computational unit treats a received message. There are two kinds of received messages, where one is observation message sent by other node while the other is neighbour message. Notice that the received observation message is different from the observation message generated in observationEvent. The former is sampled by the sensor of another node and transmitted to this node, and the latter is sampled by the sensor of this node.

In the UPPAAL model a received message is treated as follows (which refers to the generic framework):

- An observation message sent by another node is treated by first checking if this node is the destination for the message. If not, this message is ignored and a transition to **Idle** location is performed. Otherwise, the message is put into the output-queue waiting for forwarding to the next destination. For the same reason with observation message produced in observationEvent, the next destination for the received observation message is computed just before the message is sent out instead of during the treatment of the event. In addition, it is obvious that there is no need to forward the message if this node is the sink node. So, there are three transitions emitted from the location **Handle_OBS** at the right half of the automaton.
- A neighbour message is treated by first updating the neighbour view of this node at the left half of the automaton. Then, a new routing state must be computed by the function **updateNext(*id*)**. The content of this function depends on the applied energy-aware routing algorithm. After that, the next destination of this node is updated. In this treatment of neighbour message, the difference from the generic framework is that even if the change to the computational state is significant, the abstract state of this node will not be communicated to the neighbours immediately. Instead, the change to the computational state is recorded and the abstract state will be sent out when a readEnergyEvent happens.

For simplicity it is assumed that a receiveEvent is reacted instantaneously in this timed automaton by using committed states.

Reacting to readEnergyEvent

The timed automaton for the part of the treatment of readEnergyEvent can be seen in Figure 4. 12. As explained before, the variable **energyEvent[*id*]** is used to notify the computational unit that a neighbour message is required since the change to the energy state of the node is significant. Therefore, a neighbour message is generated and put into the output-queue.

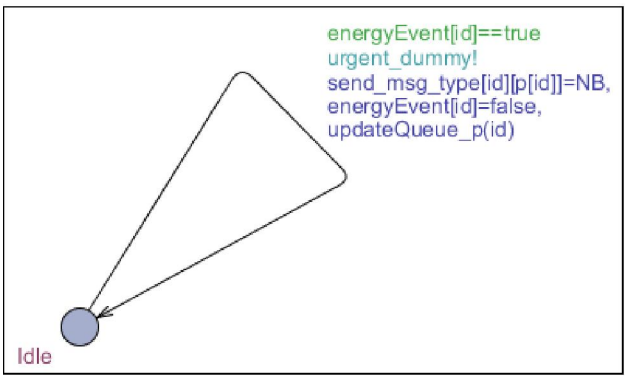


Figure 4. 12 The treatment of readEnergyEvent in computational unit

The timed automata provided in this section are for the physical components of an individual node. It is clear that they can be used for a network of nodes by giving different identities to them. The question then is how to model the relationships between nodes to build that network. The next section introduces the UPPAAL model of the communication medium, which is one of the extensions to the generic framework presented in Section 3.1.

4.3 Communication Medium

In the UPPAAL model, the communication media of the nodes are expressed by a two-dimension array **visible[*id1*][*id2*]**. As shown in Figure 4. 13, a two-dimension array determines which nodes can receive a message sent by a given node for a sample network with seven nodes in total. For example, the fourth row **visble[*3*][*id2*] = {false,true,true,false,true,true,true}** determines that the communication medium of node N_3 is $\{N_1, N_2, N_4, N_5, N_6\}$.

```
bool visible[NUM_NODES][NUM_NODES] = {{false,false,false,false,false,false,false},
                                      {true,false,true,true,true,true,true},
                                      {true,true,false,true,true,true,true},
                                      {false,true,true,false,true,true,true},
                                      {false,true,true,true,false,true,true},
                                      {false,true,true,true,true,false,true},
                                      {false,true,true,true,true,true,false}};
```

Figure 4. 13 The two-dimension array for communication medium

This two-dimension array is a kind of dynamic array, which means that the element of this array can be modified by the nodes in order to express the change of the communication media. For the same example, the communication medium of N_3 may change to $\{N_1, N_4, N_5\}$ because of the reduction of the radio signal range of N_3 caused by energy deficiency.

In the description of the communication medium presented in Section 3.1, this dynamic change relies on the physical distance between nodes and the radio signal range. There should be a model defines that the relationship between the signal strength and the residual energy level. Then, the communication medium can be updated by comparing the signal range and the physical distance between this node and the others. Incorporating this model in UPPAAL can be a future work to this Master project.

4.4 Energy Model

Another extension of the generic framework is the energy model given in Section 3.2. The energy model is partitioned into a battery and a capacitor. In the UPPAAL model, the battery is represented by an integer variable **battery[*id*]**, which is used to record the residual energy level. Moreover, the cost functions, such as **costSend** and **costRead**, are executed by this variable via reducing constant values like in Figure 4. 2 and Figure 4. 7.

In reality, the consumption of the battery is a continuous process as presented in Subsection 3.2.2. For simplicity in the UPPAAL model this continuous process is substituted by a discrete process as shown in Figure 4. 14. A continuous model for energy consumption in UPPAAL does not seem possible to achieve because a continuous progress is reached by a clock variable which can evolve with different rates, but a clock cannot progress with negative rates. Therefore, only continuous energy increase can be achieved in UPPAAL for now.

The reason why it is acceptable is that for each of the cost functions the amount of energy consumed is fixed and predictable, and also these cost functions do not happen very frequently.

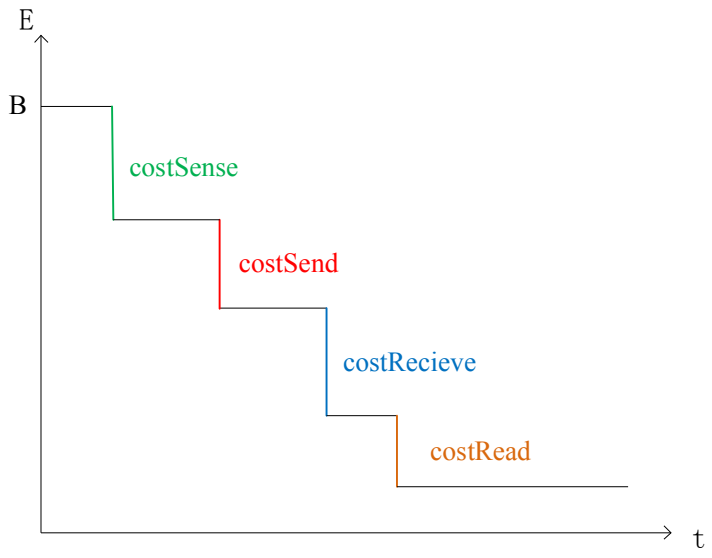


Figure 4. 14 An example of discrete consumption of the battery

On the other hand, the capacitor of the energy model is expressed by a continuous variable **capacitor[*id*]**, where the "continuous" is achieved by using a clock variable since a clock variable evaluates to a real number in UPPAAL.

One of the tasks of the capacitor is to transfer energy to the battery as introduced in Subsection 3.2.1. In the UPPAAL model, this task is incorporated in the originating of the event readEnergyEvent. Figure 4. 15 gives a new version of the timed automaton shown in Figure 4. 7 with this incorporation.

From the timed automaton we can see that every time the processor originates the event readEnergyEvent, the battery is charged by the capacitor. Since **battery[*id*]** is an integer and **capacitor[*id*]** is a clock, the operation "**N<=capacitor[*id*] && capacitor[*id*]<N+1**" needs to be done. It is used to obtain the integer part of the real number expressed by **capacitor[*id*]**. The reason for doing that is because it is not allowed to assign a clock to an integer in UPPAAL.

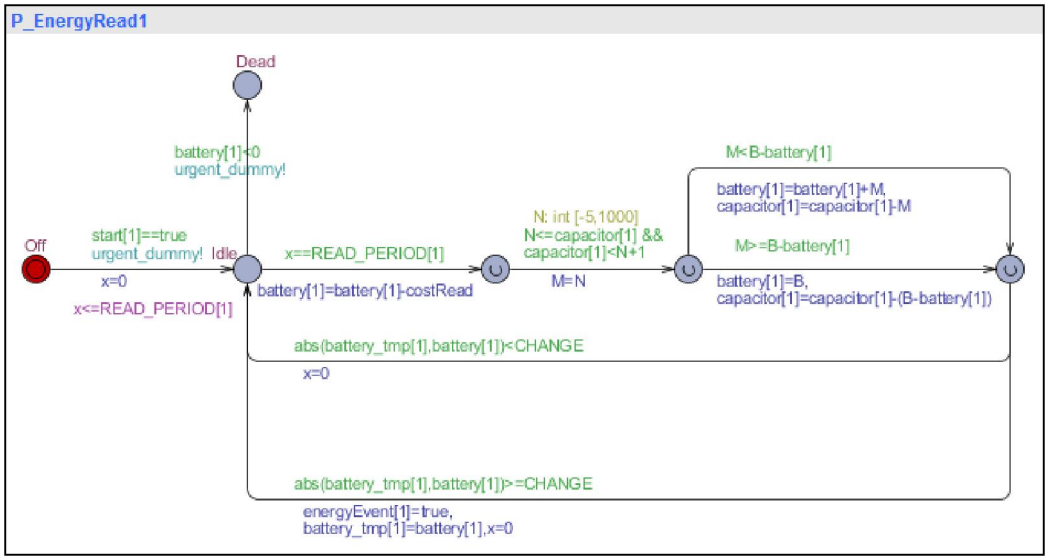


Figure 4. 15 A new version of UPPAAL model for the originating of readEnergyEvent

The other task of the capacitor is storing the energy produced by the energy harvester. To simulate the energy production in the capacitor, we must provide a clock variable which can progress with different rate, as the power production dynamically changes according to environment conditions. Fortunately, this new feature of clock is available in UPPAAL-SMC. The detailed comments to the clock rate will be given together with the environment model in next section.

4.5 Environment Model

The environment model describes the energy source for harvesting by each node. Figure 4. 16 gives the timed automaton for the solar environment model ($Harvester_1$) presented in Section 3.3.

The solar environment model is created under the support of UPPAAL-SMC. The new engine of UPPAAL-SMC supports the timed automata whose clocks can progress with different rates in different locations. This property is used to simulate the continuous energy production in capacitor. Furthermore, UPPAAL-SMC offers a user interface to specify complex problems with probability distributions. It allows us to specify the probabilities of different kind of weather conditions in the environment model. Besides, the behaviors of a model can be defined by adding distributions on the delay before a transition is taken. Those distributions are uniform if the delays are bounded, and exponential otherwise. This property makes the environment model very natural.

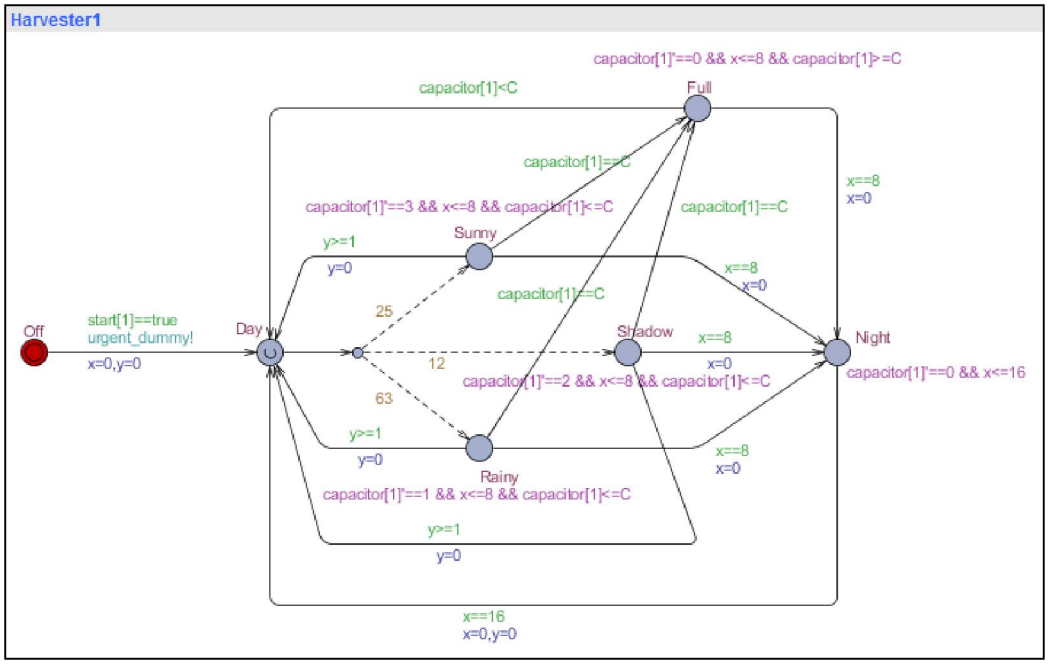


Figure 4. 16 UPPAAL model for the solar environment model

The locations **Sunny**, **Shadow** and **Rainy** in the automaton correspond to the three harvesting scenarios – clear sky, light shadow and strong shadow respectively. Considering a node deployed in Copenhagen in January, the probability of its energy harvesting under each of these scenarios can be obtained from Table 3. 1. UPPAAL-SMC offers a friendly user interface which allows specifying complex problem with probability distribution like the dotted line in the figure. Here the probability weights to these three scenarios should be specified to 25%, 12%, 63% roughly according to the table.

When the energy harvester is started, it first checks to see which kind of environment conditions it is under. There is 25% chance that the harvester works under clear sky as shown in Figure 4. 17.

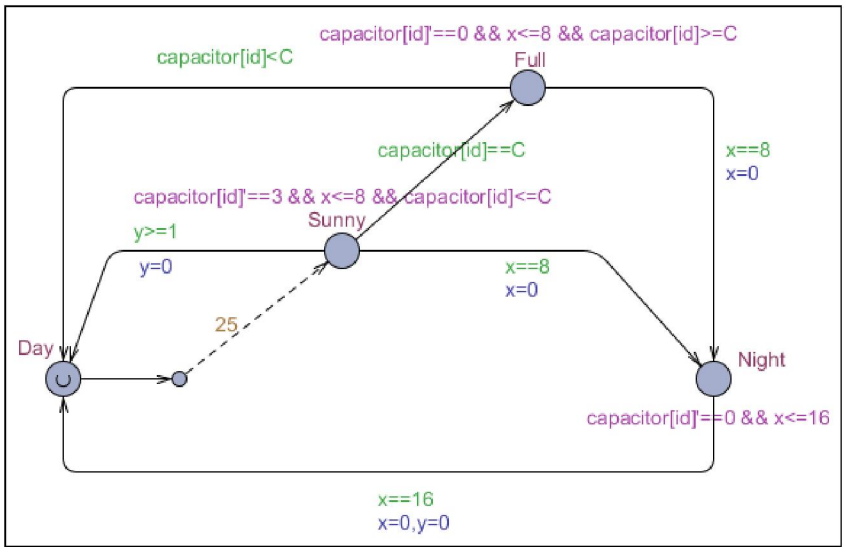


Figure 4. 17 The part of the automaton with Sunny state

If the environment goes to **Sunny** location, the power production by the harvester should be the best of all. In the timed automaton, this is done by setting the rate of the clock variable **capacitor[*id*]** to 3. The harvester can work at **Sunny** location at most for 8 time units (refers to 8 hours), and then night falls. During night, the capacitor stops charging, so the rate of the clock variable **capacitor[*id*]** is set to 0. After 16 time units (refers to 16 hours), a new day comes.

It should be noticed that the environment condition is likely to change at any time, so there is a transition that goes back to the urgent location **Day** from location **Sunny**. The probability distribution of this transition is uniform from 1 to 8 since the invariant is bounded by a clock **y**, otherwise it is exponential. If going back, the environment will check its condition again on the basis of the probability weights 25%, 12% and 63% for each location.

The capacitor may reach its capacity during its charging, so there is a location **Full** connected to **Sunny** location. At this location, the capacitor stops charging and waits for some energy consumption. As long as the capacitor is not fully charged, it will start to store harvested energy.

The locations **Shadow** and **Rainy** in the timed automaton shown in Figure 4. 16 are very similar to the explained location **Sunny**. The only difference is the clock rate applied to **capacitor[*id*]**. This change is made in order to express smaller power production under different environment conditions.

4.6 Discussion

In this chapter a formal UPPAAL model of energy harvesting aware WSNs is created on the basis of the generic modelling framework. Further formalization concerning the network communication medium and the energy harvesting capability can be found in the UPPAAL model as well. These formalized parts of the framework provide good bases for simulation and verification.

It is desirable and challenging to define a communication medium which can capture the issues of message loss and collision, but for now having a model that determines which nodes can receive a message sent by a given node is good enough.

The energy model and the environment model for realizing energy harvesting capability are made under the support of UPPAAL-SMC. With some extended properties, such as probability distributions, the energy consumption and production of a sensor node can be simulated in a very natural way.

With the formal UPPAAL model in hand, many of the implicit issues and problems of the generic framework are made clear. Concrete experiments of simulation and verification, based on this formal model, can be found in the next chapter. The full timed-automata for this UPPAAL mode are given in Appendix A.

CHAPTER 5.

Experiments: Simulation and Verification

In this chapter we will study some experiments of the WSN model presented in the UPPAAL chapter. Representing the modelling framework in the UPPAAL model provides some very useful tools and features.

- Formal explanation: The UPPAAL model of energy harvesting aware WSN provides an exhaustive and formal explanation while the modelling framework only describes the system informally and in general terms.
- Simulation: The UPPAAL simulator provides a great feature that enables intuitive examinations of possible results of a system. The simulation in UPPAAL is a cheap mean of validation which could be very useful during early modelling phases.
- Verification: Verification through model checking is the most important feature that UPPAAL gives. The verifier is to check safety and liveness properties by exploring the state-space of a system. The issue of state-space explosion, which makes the verification of large system very time-consuming, is problematic for classical model checkers. Therefore, Statistical Model Checking extension of UPPAAL is applied to solve this kind of problem in this Master thesis.

In order to present the experimental process, this chapter is divided into two parts, where one is concerned with the preliminary experiments for a simple layer network that applies the Simple DEHAR algorithm, while the other is concerned with the comprehensive experiments for a common network implementing the DEHAR algorithm.

5.1 Preliminary Experiments

In this section it will be demonstrated that the energy-aware routing algorithm Simple DEHAR can be used to increase the lifetime of a WSN. In order to do so, Simple DEHAR must be able to increase the lifetime of the key nodes which are close to the base station, as the relative short lifetime of the network is usually caused by the drain of such nodes.

To concentrate on the effect of energy awareness in the first step, the capability of energy harvesting of the WSN model is omitted in these preliminary experiments.

The considered network is given in Figure 5. 1. The reason why this network only has five nodes is that the preliminary experiments are conducted in classical UPPAAL, and the classical model checker can handle at most five such nodes due to state-space explosion. This is not surprising because in the UPPAAL model each node has several parallel automata, and each automaton has several states, as a result the required state-space is very large.

The considered network has two nodes spatially close to the base station N_0, and one of them (N_1) is in problematic state with an energy deficit. We will analyze the ability of Simple DEHAR algorithm to extend the lifetime of this problematic node.

Apart from the Simple DEHAR algorithm a simplified version of the Directed Diffusion (DD) algorithm is also implemented. This simplified version of DD implements the shortest distance to the base station for every node in the network, so it is not energy-aware routing algorithm. This is used to show the difference between DD and the Simple DEHAR.

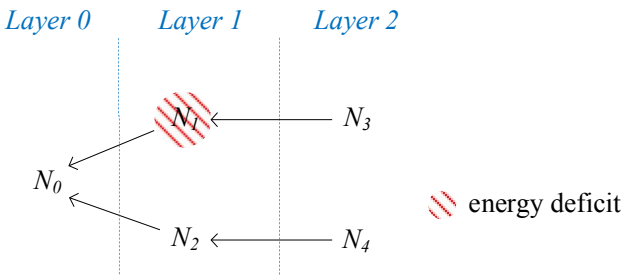


Figure 5. 1 A simple layer network structure

The communication medium must be defined for the preliminary experiments. It is assumed that a node can communicate with the nodes in its adjacent layer. Moreover, the arrows given in the figure illustrate the initial next destination of each node.

The energy model is simplified to an individual battery since the energy harvesting capability is skipped in the preliminary experiments. Except for the problematic node N_1, the batteries of all nodes are fully charged initially.

5.1.1 Simulation

Two simulations are conducted in classical UPPAAL model during the preliminary experimental stage, one with DD algorithm and another with Simple DEHAR algorithm. Table 5. 1 shows the parameters of the nodes which include the frequency of observations, duty-cycle of the devices and energy consumption of events.

		unit	N_1	N_2	N_3	N_4
Sensor	**Active Duration**	s	10	10	10	10
	Period	s	110	105	16	63
	costSense	J	30	30	30	30
Receiver	**Active Duration**	s	5	5	5	5
	Period	s	10	10	10	10
	costReceive	J	20	20	20	20
Transmitter	**costSend**	J	50	50	50	50
Processor	**costRead**	J	5	5	5	5

Table 5. 1 Parameters used in simulations

It should be noticed that the frequency of observations sampled by node N_3 (10/16) is set larger than the others. This is done because the next destination of N_3, i.e. N_1, is the node close to the base station, which will be activated much more frequently in practice. Here the large frequency of observations in N_3 is used to arouse the frequent activation of N_1.

Figure 5. 2 gives the left side of the simulator – including enabled transitions, simulation trace and value of variables. It can be seen that initially the next destinations of node N_3 and N_4 are N_1 and N_2 respectively (**next[3]=1** and **next[4]=2**), and only the battery of node N_1 is not fully charged (**battery[1]=500**).

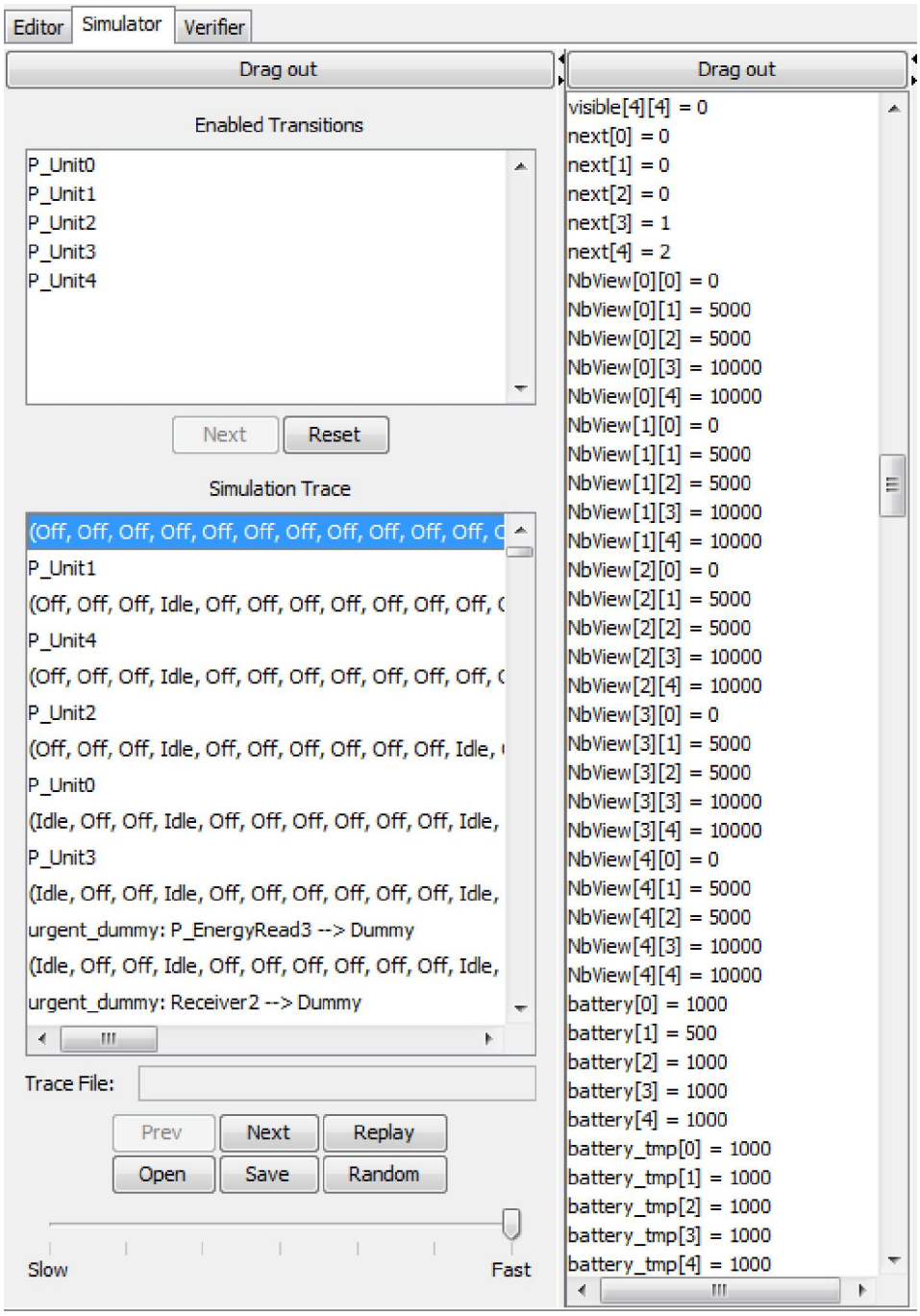


Figure 5. 2 Left side of UPPAAL simulator

Energy awareness makes a difference

A randomized simulation implementing Simple DEHAR algorithm is shown in Figure 5. 3. The figure shows the time (**t = 135**) when the problematic node dies.

Through the simulation it can be seen that the problematic node N_1 is avoided for a period of time, because the next destinations of node N_3 is changed to N_2 (**next[3]=2**). When the

residual energy of N_2 is less than N_1, the next destination of N_3 is changed to N_1 back.

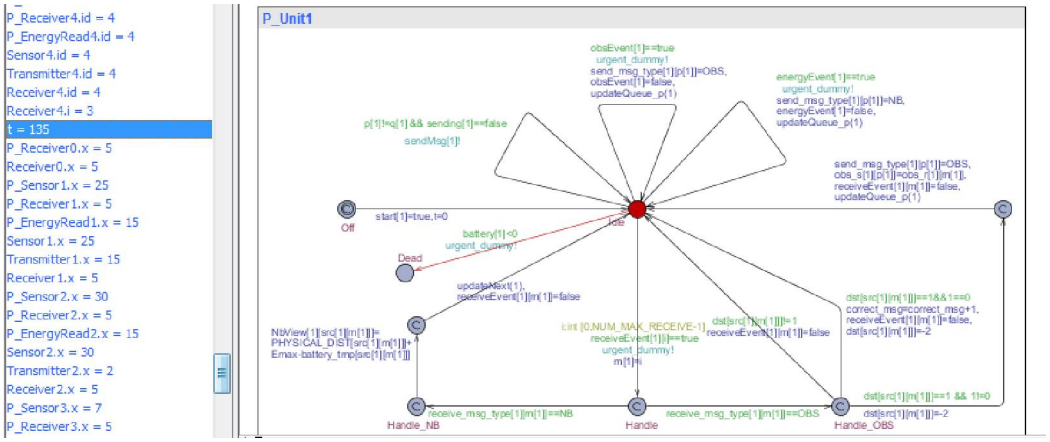


Figure 5. 3 Results of a randomized simulation with Simple DEHAR algorithm

To demonstrate the effect of energy awareness, another randomized simulation which applies the simplified DD algorithm is conducted to compare with. The parameters used in these two simulations are exactly same. The result of the second simulation is given in Figure 5. 4.

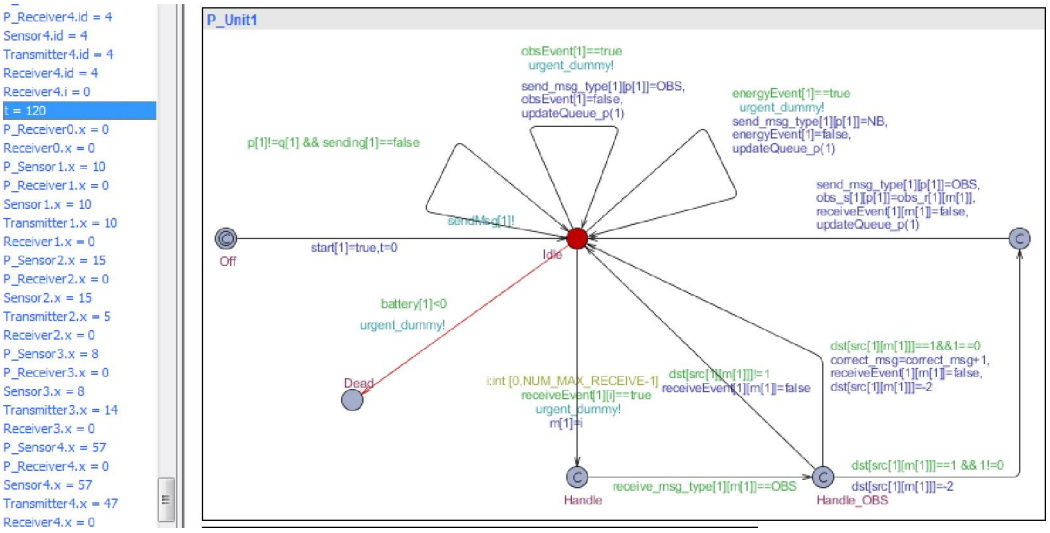


Figure 5. 4 Results of a randomized simulation with DD algorithm

The time (**t**) when the processor of node N_1 goes to **Dead** location is **120** seconds. Therefore, the energy aware routing algorithm Simple DEHAR extends the lifetime of the problematic node. The reason why the death of N_1 is only postponed for 15 seconds is that there is only one node transferring observations to the base station through N_1. It is not difficult to imagine that if the network has a large number of nodes, which need N_1 to be intermediate destination to route data, the energy of N_1 will be rapidly drained in the DD case.

5.1.2 Verification

As mentioned before, verification is the main goal to represent the modelling framework in UPPAAL. The verifier of UPPAAL can check invariant and reachability properties of a system.

The verification conducted in the preliminary experiments is for the network shown in Figure 5. 1. Some queries (i.e. system requirements) of the network with Simple DEHAR algorithm are verified by the model checker of UPPAAL. One of the most common liveness queries is:

E <> deadlock

This query returns ***Property is satisfied***, since a deadlock is detected when all of nodes in the network are drained of energy.

Another valuable liveness query is:

E <> Sensor2.Sensing && Transmitter2.Sending && Receiver2.Receiving

This query returns ***Property is satisfied***, so it is demonstrated that some physical components of a node – the sensor, the transmitter and the receiver can work in parallel.

An important safety query checked in verification is:

A[] deadlock imply P_Unit0.Dead && P_Unit1.Dead && P_Unit2.Dead && P_Unit3.Dead && P_Unit4.Dead

This query returns ***Property is satisfied***, which means the system will deadlock only when all of nodes are in **Dead** state. Therefore, the verified system is safe.

Another safety query is

*A[] deadlock imply correct_msg*10/total_msg>8*

This query returns ***Property is satisfied***, which means the number of successful communication messages is more than 80% of the number of all produced messages. This query can be used to measure the system performance.

Figure 5. 5 gives a graph of the overview of these queries after verification, the green light behind each query illustrates that property is satisfied.

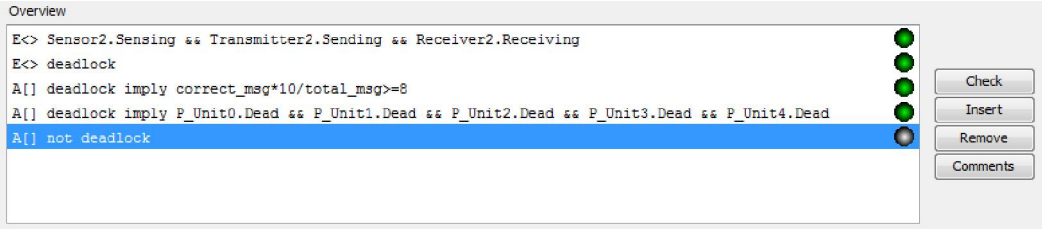


Figure 5. 5 The overview of the queries

5.2 Comprehensive Experiments

This section introduces some comprehensive simulations and verifications conducted during the further experimental stage. The considered network is shown in Figure 5. 6. The network has one problematic node N_3, which has energy deficiency in its battery. We will study the ability of the energy harvesting aware routing algorithm DEHAR to cope with this problematic node using simulation and verification.

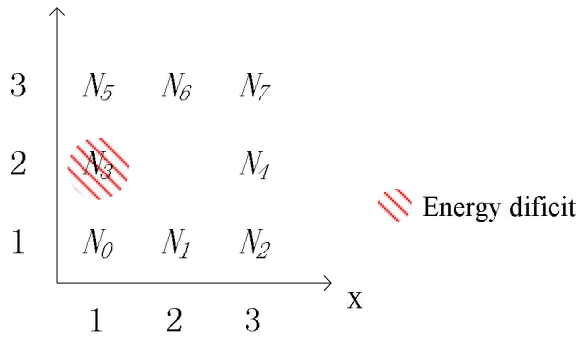


Figure 5. 6 A network structure with illustrating problematic node

The communication medium assumes that a node can communicate with its immediate horizontal and vertical neighbour. The initial next destination of each node is shown in Figure 5. 7 by arrows.

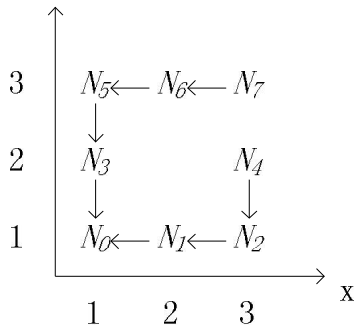


Figure 5. 7 A network structure with illustrating active routes

The energy model is partitioned into a battery and a capacitor as introduced in Section 3.2 and 4.4. The batteries of all nodes except the problematic one are fully charged initially. The work efficiency of the capacitors is biased towards the environment model, but in the beginning all the capacitors are empty.

5.2.1 Simulation

The simulation implementing the DEHAR algorithm for the considered network is conducted without energy harvesting, since the simulator in UPPAAL is symbolic and cannot handle a clock variable with different progress rate. However, the simulator helps to check the correctness of the algorithm. Furthermore, by simulation some potential problems, which are not considered when establish the modelling framework, are detected. On the other hand, the capability of energy harvesting of the DEHAR algorithm can be checked in verification with the support of UPPAAL-SMC extension as shown in next subsection.

This simulation uses very approximate parameters in Table 5. 1, only the number of nodes and the frequency of observations have a little bit of difference. But the frequency of observations is configured in accordance with the same approach: nodes far away from the base station have more observations than those close to the base station. The aim of this approach is to simulate that a lot of observations are collected by many remote nodes in the network. So, it provides the probability to expand the network with more nodes.

As was the case with the preliminary experiments, two simulations are conducted in the comprehensive experiments, one is implementing the DEHAR algorithm, and another is implementing the simplified DD algorithm. From these two simulations, it can be seen that energy awareness makes difference, because in DD case the problematic node N_3 dies at **114** seconds while in DEHAR case the node dies at **155** seconds. This result demonstrates that the gain by using energy aware routing algorithm exceed the energy required by executing the algorithm if the configuration is correct and delicate.

Figure 5. 8 presents two new consistent state of the network produced when implementing the DEHAR algorithm in simulation. Figure 5. 8(a) is the routing trend when the change caused by energy deficit of N_3 has settled. Figure 5. 8(b) displays another routing trend constructed by the energy deficit of N_6 due to its sensor's frequent activation.

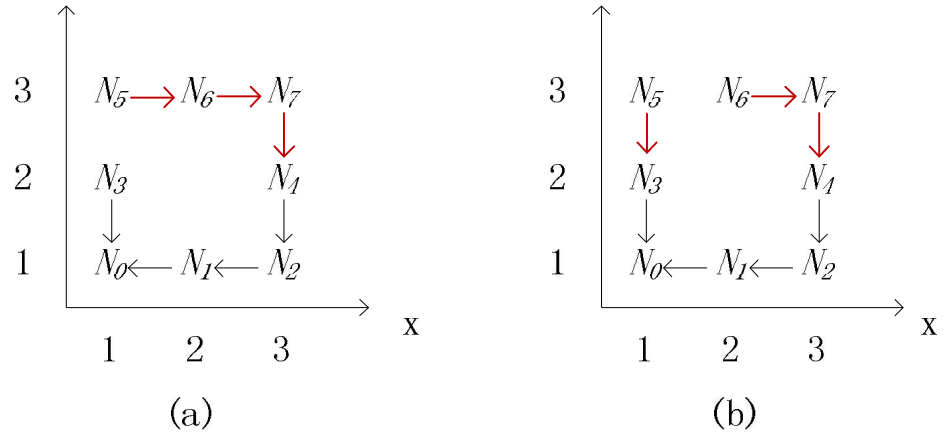


Figure 5. 8 New consistent state of the network. (a) shows the first new consistent state caused by energy deficit of N_3. (b) shows a followed new consistent state caused by large energy consumption on N_6 due to frequent sensing.

Notice that this simulation is conducted under an assumption that all neighbour messages are guaranteed to be successfully communicated between nodes. Even though the receiver of a node works in duty-cycle mode, it must be activated if there is another node transmitting neighbour message to it.

The reason for making this assumption is that if a neighbour message can be lost, the network may never reach a consistent state, and therefore a lot of observations will be blocked in the network and never reach the base station.

Without this assumption, this potential problem can be detected in the simulation. When the routing trend shown in Figure 5. 8(a) is constructed, the first neighbour message sent from N_5 to N_6 may be lost, and then these two nodes will see each other as the next destination as shown in Figure 5. 9. As a result, the observations sampled by N_5, N_6 and N_7 will be blocked until another neighbour message is communicated successfully between N_5 and N_6. However, this successful communication may never come. Therefore, it is necessary to guarantee that all of neighbour messages are communicated correctly in this WSN model.

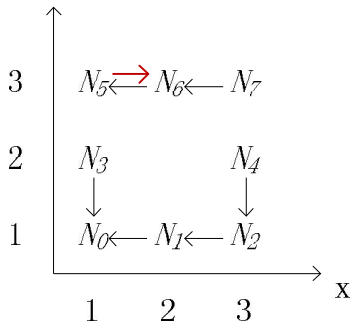


Figure 5. 9 The routing trend of the network when N_5's neighbour message is lost

This assumption is rational to be made because some low-level protocols applied in energy harvesting aware WSNs are capable to deal with these issues. An introduction of an example of such low-level protocols can be found in Chapter 6, and incorporating a low-level protocol to the UPPAAL model would be desirable in future development.

5.2.2 Verification

The verifications conducted in comprehensive experiments are based on the support of UPPAAL-SMC extension, which enables the efficient analysis of system properties of priced network by statistical model checking approach. UPPAAL-SMC offers a friendly user interface that allows a user to specify complex and undecidable problems as well as get feedback in the form of probability distributions.

The idea of SMC is to simulate a system for finitely many runs and then uses a statistical algorithm to get estimates for the system. In contrast with a classical verifier, such a simulation-based solution does not guarantee a result with 100% confidence. However, it is possible to bound the system properties by time or cost. Moreover, simulation-based method is far less memory and time consuming, so the problem of state-space explosion can be avoided. This method can be seen as a trade-off between simulation and formal verification.

With the support of UPPAAL-SMC, some undecidable problems in the proposed WSN model can be specified by probability distributions, such as weather changes in environment model and clock skew between nodes. Besides, a clock variable progressing with different rate is available in UPPAAL-SMC, which makes the continuous and dynamic energy harvesting achieved as introduced in Section 4.5.

Figure 5. 10 gives the result of the simulation-based verification for the lifetime of the problematic node N_3. The query is:

Pr[t<=200] (<> P_Unit3.Dead)

and the result is a probability interval [0.711518, 0.811518], which means that the probability of node N_3's death at 200 seconds is about 75%. This estimate is obtained from 738 runs of the simulations.

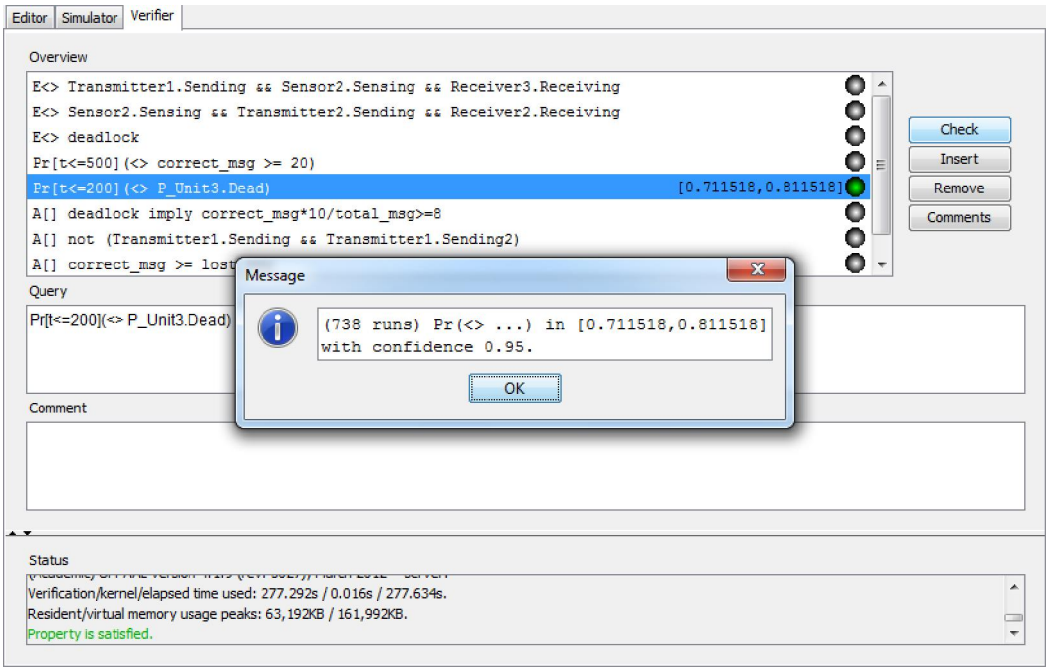


Figure 5. 10 Result of the simulation-based verification for the lifetime of the problematic node

Figure 5. 11 gives a plot of the probability distribution of the verified query. From this graph we can get the probabilities of the problematic node's death at different time. And also an average estimate probability is given in the graph.

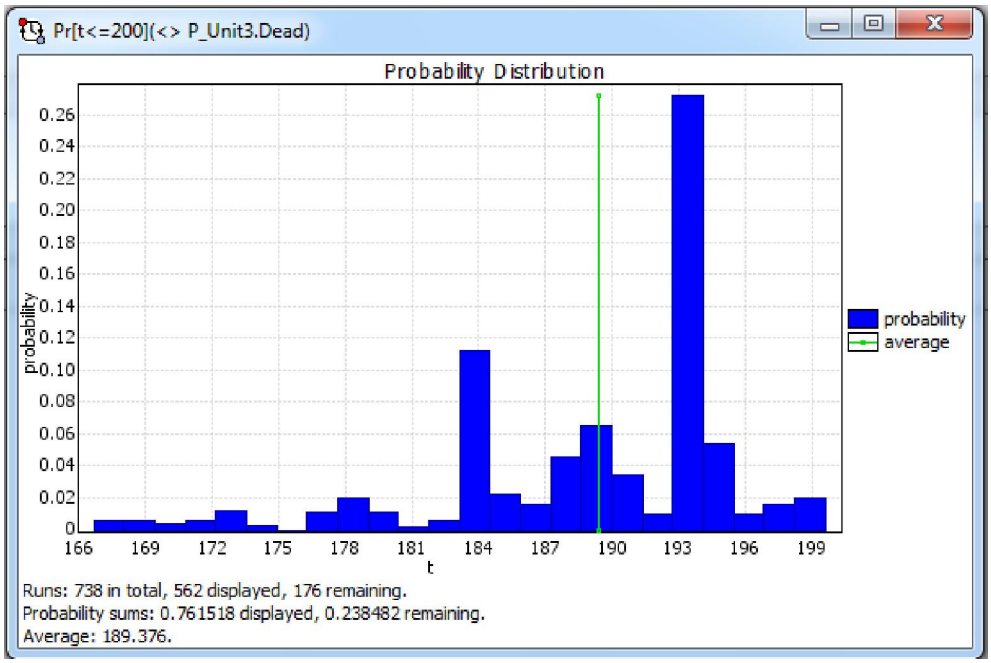


Figure 5. 11 The plot of the probability distribution of the verified query

Apart from the proposed query, some qualitative check and comparison check are also done in the comprehensive experiments. Two examples of the queries are given below:

$$Pr[t<=500](<> correct\_msg >= 20) >= 0.5$$

and

$$Pr[t<=200] (<> P\_Unit3.Dead) >= Pr[t<=200] (<> P\_Unit1.Dead)$$

The first query is to check if the probability that there are more than 20 observations transferred correctly to the base station after 500 seconds is at least 50%, and the result is **true**.

The second one compares the probability of node N_1 and N_3's death at 200 seconds, and the result is **true** because N_3 dies before N_1's death in most of the simulations.

All these three kind of checks rely on some statistic algorithm to estimate the properties of the system by observing 738 runs of the simulations.

In further development, the possibility of verification of other properties of the system would be desirable.

5.3 Discussion

The results of the simulation from both the preliminary and the comprehensive experiments show that energy awareness helps to prolong the lifetime of critical nodes in the network. The proposed algorithm is able to find sustainable path from any node to the base station.

The presented verification in comprehensive experiments involves energy harvesting capability for each sensor node. As we can see at Figure 5. 11, the drain of energy in critical node can be further postponed. These results validate that energy harvesting aware algorithm increase the lifetime of the network as a whole.

Although the results are encouraging, there are many possible improvements and extensions to the experiments. One of which is expansion of the considered network with more sensor nodes. The other can be testing the applicability of the UPPAAL model with a broad range of energy harvesting aware routing algorithms.

CHAPTER 6.

Extension

This extension chapter gives a brief introduction of a low-level protocol designed for energy harvesting WSNs. It is an On Demand Medium Access Control (ODMAC) protocol [35]. This protocol is able to support individual duty-cycles for nodes with different energy profile. It explains why an assumption that all neighbour messages are successfully communicated between nodes can be made in the presented high-level UPPAAL model and related experiments.

In general, ODMAC has the following key features. First, the communication is occurred in an on demand manner, in the sense that the transmitter does not send a frame unless the receiver asks for it. This way the nodes are minimizing the energy waste while maintaining a communication scheme that can support individual duty-cycles. Secondly, the protocol provides the network administrator a tool to favor application requirements. Finally, ODMAC incorporates an optimistic forwarding scheme to decrease the end-to-end delay.

The basic idea behind ODMAC is that each receivers periodically broadcast a beacon to inform transmitters when it is ready to accept packet. All nodes are listening to the channel so that receive an appropriate beacon for sending data packet. By using this method, the receivers spend energy on broadcasting the beacon frame instead of idle listening in traditional way. The transmitter waste energy when it is waiting for a beacon, but it is not significant in low traffic network.

Figure 6. 1 shows the basic communication between an ODMAC transmitter and an ODMAC receiver. The node A and node B represents the receiver and the transmitter respectively. Suppose that node B wants to transmit a data packet to node A, it listens the channel for the beacon so that it can start sending. Node A is in sleeping state at first, after that it wakes up and listens to channel for period T_{IFS}, then it broadcasts beacon if the channel is free during defined

time, otherwise node A returns to sleeping state. A waiting time T_{TX} is initiated after beacon is transmitted, while node A waits for the incoming packet. If there is no incoming packet during the time T_{TX}, node A returns to the sleeping state. For node B, if receives the beacon and identifies it originates from node A, it back off a random number of time slot T_{SLOT} before transmitting. Unless the channel remains free, node A quits this beacon and starts listening next one. If the channel is free, node B initiates the transmission, and it will return to a sleeping state when the transmission is successfully finished; unless it has more queued packets. After each successful transmission, node A does not return to a sleeping state. Instead, it transmits a new beacon immediately. This design solves the contention issue and helps the transmitters with queued packet.

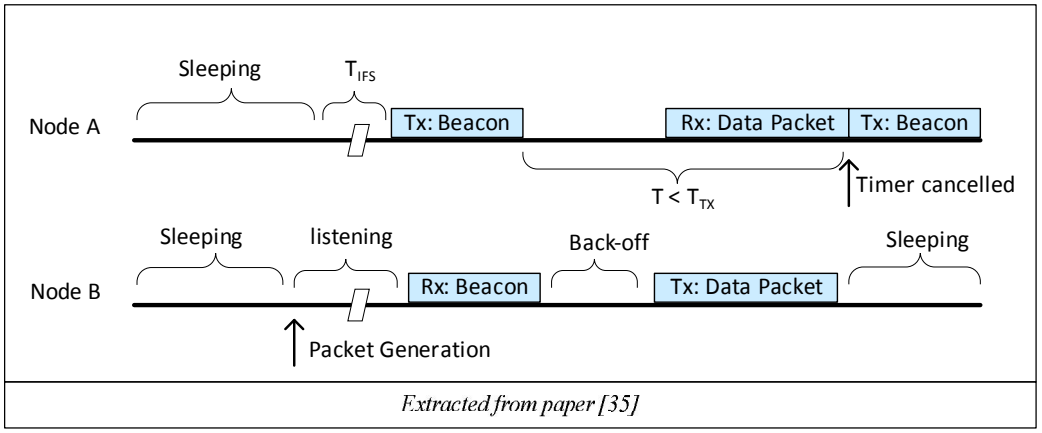


Figure 6. 1 Typical communication between a transmitter and a receiver in ODMAC

The integrated channel sensing timers avoid and handle a series of potential errors. One situation is that a node transmits a beacon while another node has already transmitted one and is waiting to receive a packet. The communication between nodes should not be interrupted. T_{IFS} is designed for solving this problem. ODMAC guarantees that communication between the nodes will never be interrupted by ensuring $T_{IFS} > T_{TX}$. Second potential error occurs if two transmitters are waiting for the same beacon, packet collision may happen. This is solved by the random number of the backing off time. Another potential error occurs if two neighboring nodes transmit their beacons exactly at the same time, the beacon will collide. However, this problem still can be handled by the protocol. Therefore, eventually the receiver will go to the sleeping state and the transmitters will keep state of waiting for the next beacon.

With this low-level protocol, the energy harvesting WSNs is able to support individual duty-cycles for sensor nodes. The correct communication for neighbour messages is also guaranteed, so that the best energy-wise routes can be found for each data transfer from a node to the base station. Furthermore, the ODMAC protocol is designed with the consideration of energy consumption. It is therefore a very appropriate solution for low-level matters in Energy harvesting aware WSNs.

CHAPTER 7.

Conclusion

This thesis presents a formal UPPAAL model aimed at explaining and analyzing Wireless Sensor Networks with energy harvesting capabilities. The model is established on the basis of a generic modelling framework which can be used to study and analyze a broad range of energy harvesting aware WSNs. The framework comprises of a conceptual basis as well as an operational basis for two energy harvesting based networks. One of these networks models is based on DD, i.e. it supports energy harvesting; but the routing is not energy aware. The other network model is based on the energy harvesting aware routing protocol DEHAR.

In order to formalize the conceptual modelling framework, some details concerning the network communication medium and the environment conditions of sensor nodes were given as extensions to the generic framework, and this is another contribution of this thesis. The proposed UPPAAL model is based on the description of these extensions.

The full formal model not only provides a thorough explanation of the system while the framework only describes informally, but also supports simulation and verification using UPPAAL model checking. Modelling, simulation and verification are easily performed with UPPAAL graphical user interface.

Using classical UPPAAL, examples of simple and small Wireless Sensor Networks can be simulated and verified. The results show that energy awareness can extend the lifetime of the critical nodes, thus the whole network. The problem of state-space explosion limits the number of sensor nodes of the tested network. However, this problem can be solved with UPPAAL-SMC.

UPPAAL-SMC is a kind of simulation-based approach which is capable of handling complex and undecidable problems without consuming too much time and memory. In the proposed

formal model, probability distributions specification and clock with different rates are the used properties what UPPAAL-SMC offers. These properties make the model very natural and give the possibility to experience with larger networks.

Verification in UPPAAL-SMC provides evidence as to the correctness of the models. It further demonstrates that energy awareness can extend the lifetime of the network. Besides, with support of UPPAAL-SMC, energy harvesting in each sensor node is added to the simulation and verification.

The project has been successful in providing a formal model that not only gives results to the domain but also can be used as a basis in further development, such as validation of the applicability of the model, and implementation of the WSNs on a platform.

References

[1] Michael R. Hansen, Mikkel Koefoed Jakobsen and Jan Madsen, "A Modelling Framework for Energy Harvesting Aware Wireless Sensor Networks" in *Sustainable Energy Harvesting Technologies - Past, Present and Future,* 2011, pp. 3-24.

[2] W. Michael, T. D. Klaus, H. Kester, B. Graeme, "Theoretical and practical aspects of military wireless sensor networks", *Journal of Telecommunications and Information Technology,* 2008, pp. 37-45.

[3] Y. Ma, M. Richards, M. Ghanem, Y. Guo, J. Hassard, "Air pollution monitoring and mining based on sensor grid in London", *Sensors 2008, 8,* 3601-3623.

[4] Solobera, Javier, "Detecting forest fires using wireless sensor networks with Waspmote", *Libelium Comunicaciones Distribuidas S.L.,* 2010.

[5] S. Victor, C. Bor-rong, L. Konrad, Thaddeus R. F. F., W. Matt, "Sensor Networks for Medical Care", *Technical Report TR-08-05, Division of Engineering and Applied Sciences, Harvard University,* 2005.

[6] A. Teemu, V. Reino, E. Mohammed, "Greenhouse monitoring with wireless sensor network", *Mechatronic and Embedded Systems and Applications,* 2008, pp. 403-408.

[7] T. Jorge, V. J. Fernando, F. M. Joao, "Application of Wireless Sensor Networks to Automobiles", *Measurement Science Review,* 2008.

[8] Mikkel Koefoed Jakobsen, Jan Madsen, and Michael R. Hansen, "DEHAR: A Distributed Energy Harvesting Aware Routing Algorithm for Ad-hoc Multi-hop Wireless Sensor Networks", *Proceedings of the 2010 IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks,* 2010, pp. 1-9.

[9] L. A. Bush, C. D. Carothers, and B. K. Szymanski, "Algorithm for Optimizing Energy Use and Path Resilience in Sensor Networks", *Wireless Sensor Networks,2005. Proc. Of the Second European Workshop on,* 2005, pp. 391-396.

[10] D. J. Vergados, N. A. Pantazis, and D. D. Vergados, "Energy-efficient route selection strategies for wireless sensor networks", *Mob. Netw. Appl.,* 2008, pp.285-296.

[11]J. Faruque and A. Helmy, "Gradient-based routing in sensor networks", *SIGMOBILE Mob. Comput. Commun. Rev.,* 2003, pp. 50-52.

[12] B. Zhang and H. T. Mouftah, "Adaptive Energy-Aware Routing Protocols for Wireless Ad Hoc Networks", *Proc. Of the First Intl. Conf. on Quality of Service in Heterogeneous*

Wired/Wireless Networks, 2004, pp. 252-259.

[13] H. Hassanein and J. Luo, "Reliable Energy Aware Routing in Wireless Sensor Networks", *Dependability and Security in Sensor Networks and Systems,* 2006, pp. 54-64.

[14] R. C. Shah and J. M. Rabaey, "Energy aware routing for low energy ad hoc sensor networks", *Wireless Communications and Networking Conf., 2002,* 2002, pp. 350-355.

[15] R. P. Mann, K. R. Namuduri, and R. Pendse, "Energy-Aware Routing Protocol for Ad Hoc Wireless Sensor Networks", *EURASIP Journal on Wireless Communications and Networking*, 2005, pp. 635-644.

[16] M. S. D., M. D. C. F., B. R. I., and F. A. O., "A centralized energy-efficient routing protocol for wireless sensor networks", *IEEE Commun. Mag.*, 2005, pp. 8-13.

[17] C. Ma and Y. Yang, "Battery-aware routing for streaming data transmissions in wireless sensor networks", *Mob. Netw. Appl.,* 2006, pp. 757-767.

[18] J. Xu, B. Peric, and B. Vojcic, "Performance of energy-aware and link-adaptive routing metrics for ultra wideband sensor networks", *Mob. Netw. Appl.,* 2006, pp. 509-519.

[19] T. Voigt, H. Ritter, and J.Schiller, "Solar-aware Routing in Wireless Sensor Networks", *Intl. Workshop on Personal Wireless Communications,* 2003, pp. 847-852.

[20] T. Voigt, A. Dunkels, J.Alonso, H. Ritter, and J.Schiller, "Solar-aware clustering in Wireless Sensor Networks", *IEEE Symp. on Computers and Communications,* 2004, pp. 238-243.

[21] J. Islam, M. Islam, N. Islam, "A-sLEACH: An Advanced Solar Aware Leach Protocol for Energy Efficient Routing in Wireless Sensor Networks", *International Conference on Networking*, 2007, p. 4.

[22] E. Lattanzi, E. Regini, A. Acquaviva, and A. Bogliolo, "Energetic sustainability of routing algorithms for energy-harvesting wireless sensor networks", *Comput. Commun.*, 2007, pp. 2976-2986.

[23] L. Lin, N. B. Shroff, and R. Srikant, "Asymptotically optimal energy-aware routing for multi-hop wireless networks with renewable energy sources", *IEEE/ACM Trans. Netw.,* 2007, pp. 1021-1034.

[24] K. Zeng, K. Ren, W. Lou, and P. J. Moran, "Energy-aware geographic routing in lossy wireless sensor networks with environmental energy supply", *Pro. of the 3rd Intl. Conf. on Quality of Service in Heterogeneous Wired/Wireless Networks*, 2006, p. 8.

[25]X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks", *Pro. of the 4th Intl. Symposium on Information Processing in Sensor Networks,* 2005.

[26] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks", *ACM Trans. Embed. Comput. Syst.,* 2007, p. 32 .

[27] P. Corke, P. Valencia, P. Sikka, T. Wark, and L. Overs, "Long-duration solar-powered wireless sensor networks", *Pro. of the 4th Workshop on Embedded Networked Sensors,* 2007, pp. 33-37.

[28] F. Simjee and P. H. Chou, "Everlast: long-life, supercapacitor-operated wireless sensor node", *Pro. of the 2006 Intl. Symposium on Low Power Electonics and Design,* 2006 pp. 197-202.

[29] D. P. A. Kansal and M. Srivastava, "Performance Aware Tasking for Environmentally Powered Sensor Networks", *ACM Joint Intl. Conf. on Measurement and Modeling of Computer Systems*, 2004.

[30] C. Moser, L. Thiele, L. Benini, and D. Brunelli, "Real-Time Scheduling with Regenerative Energy", *Pro. of the 18th Euromicro Conf. on Real-Time Systems,* 2006 pp. 261-270.

[31] Rajeev Alur and David L. Dill Gerd. "A Theory of Timed Automata". *Theoretical Computer Science, 126(2):183–235,* 2004.

[32] Gerd Behrmann, Alexandre David, and Kim G. Larsen. "A Tutorial on UPPAAL 4.0". *Lecture Notes in Computer Science, 3185:200–236,* 2006.

[33] Peter B., Alexandre D., Kim G. L., Marius M., Danny B. P., Axel L., Zheng W.. "UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata". *QAPL 2012, EPTCS 85,* 2012, pp. 1-16.

[34] Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J. & Silva, F. "Directed Diffusion for Wireless Sensor Networking", *IEEE/ACM Transactions on Networking 11(1):* 2-16.

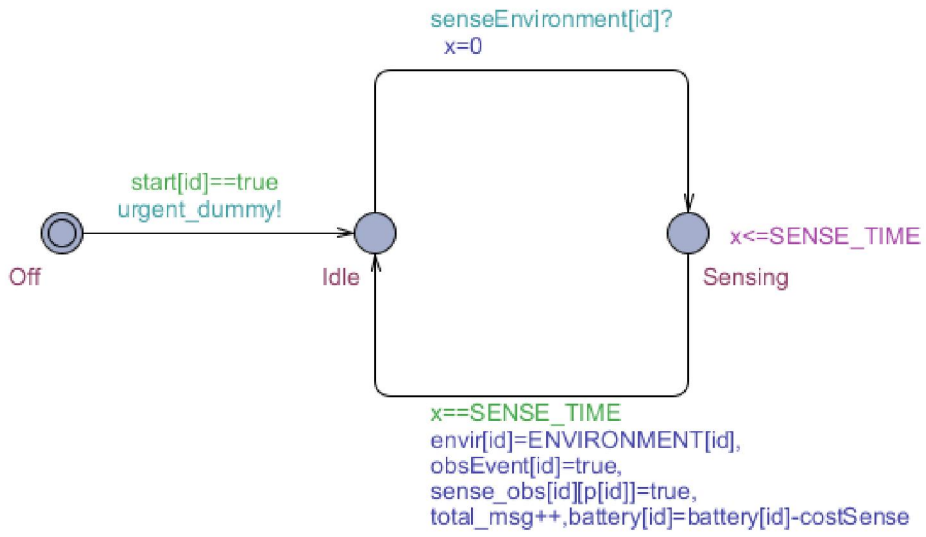
[35] Xenofon Fafoutis and Nicola Dragoni, "An On-Demand MAC Protocol for Energy Harvesting – Wireless Sensor Networks", *Pro. of the 8th ACM Symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks,* 2011, pp. 49-56.

Appendix A

Full UPPAAL Model

In the following, the full UPPAAL model specified in Chapter 4 is presented. This is done by giving each timed-automaton and their local declarations, the global declarations and the system declarations.

Sensor



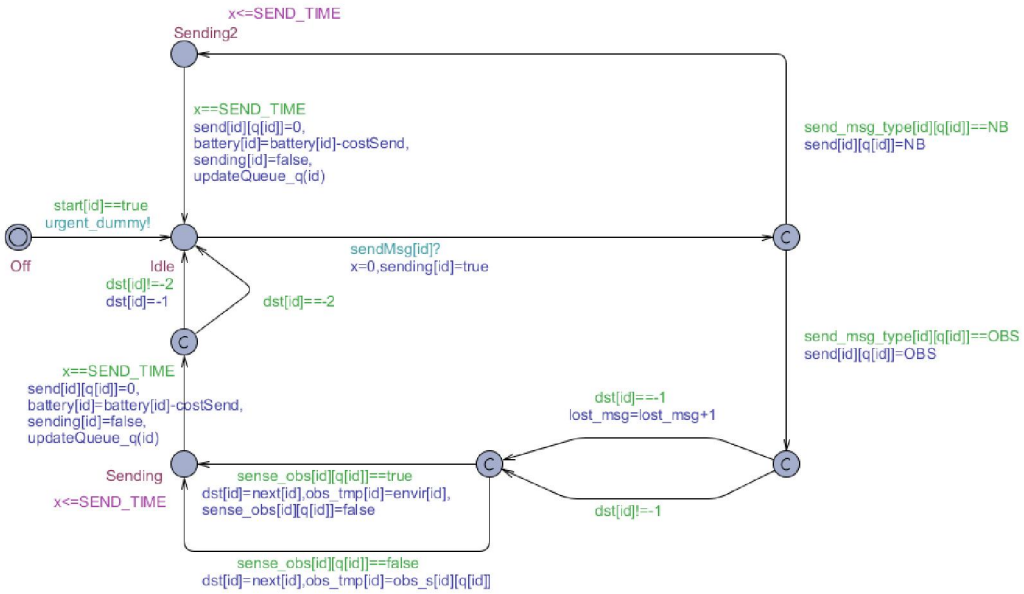
Local declarations for Sensor

clock x;

const int SENSE_TIME = 10;

const int costSense = 30; *//costSense*

Transmitter



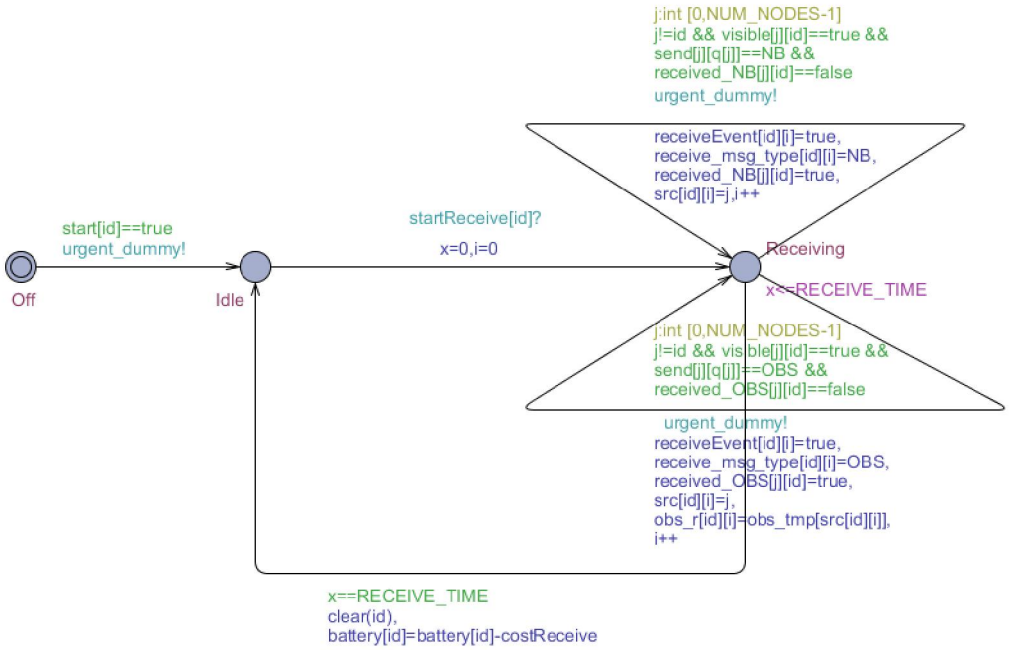
Local declarations for Transmitter

clock x;

const int SEND_TIME = 10;

const int costSend = 50; *//costSend*

Receiver



Local declarations for Receiver

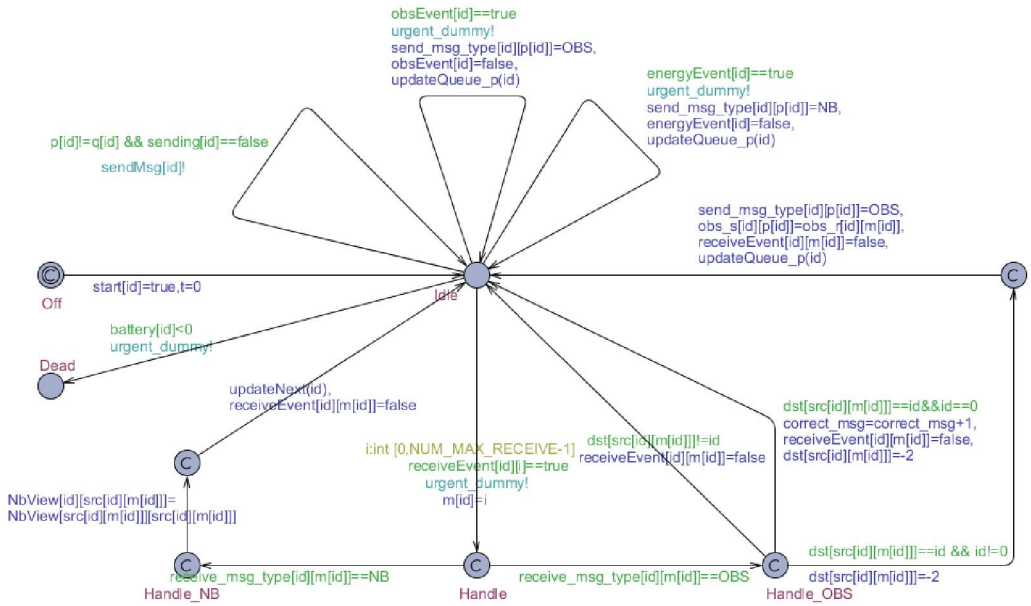
clock x;

const int RECEIVE_TIME = 5;

const int costReceive = 20; //costReceive

int i;

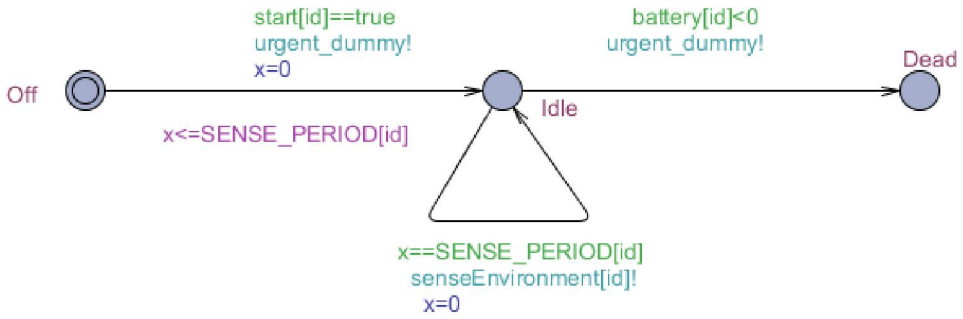
P_Unit



Local declarations for P_Unit

none

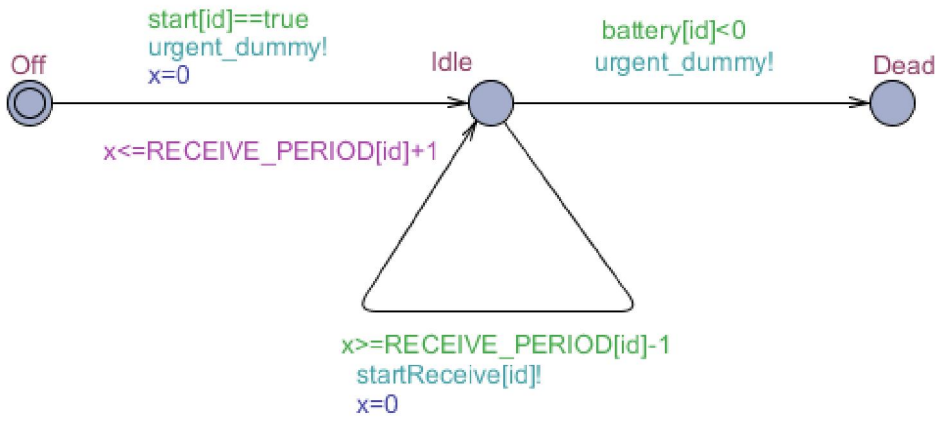
P_Sensor



Local declarations for P_Sensor

clock x;

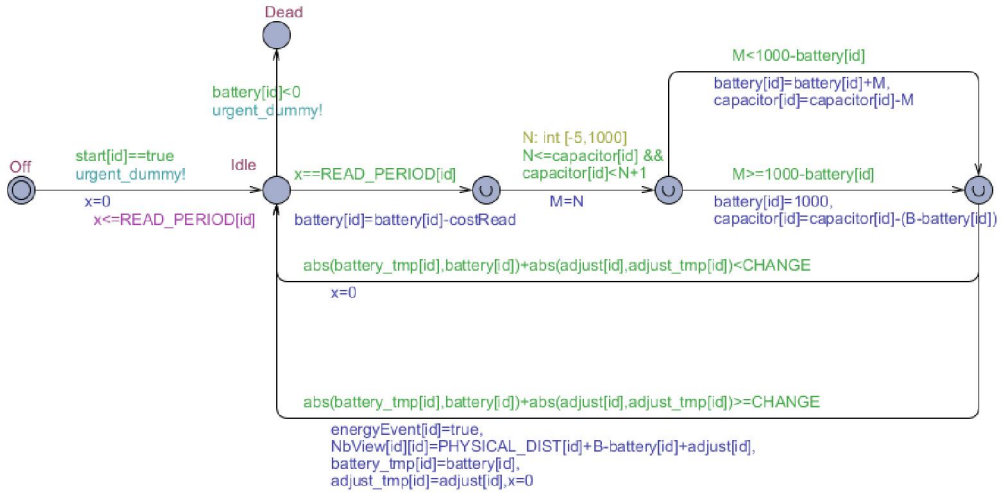
P_Receiver



Local declarations for P_Receiver

clock x;

P_EnergyRead



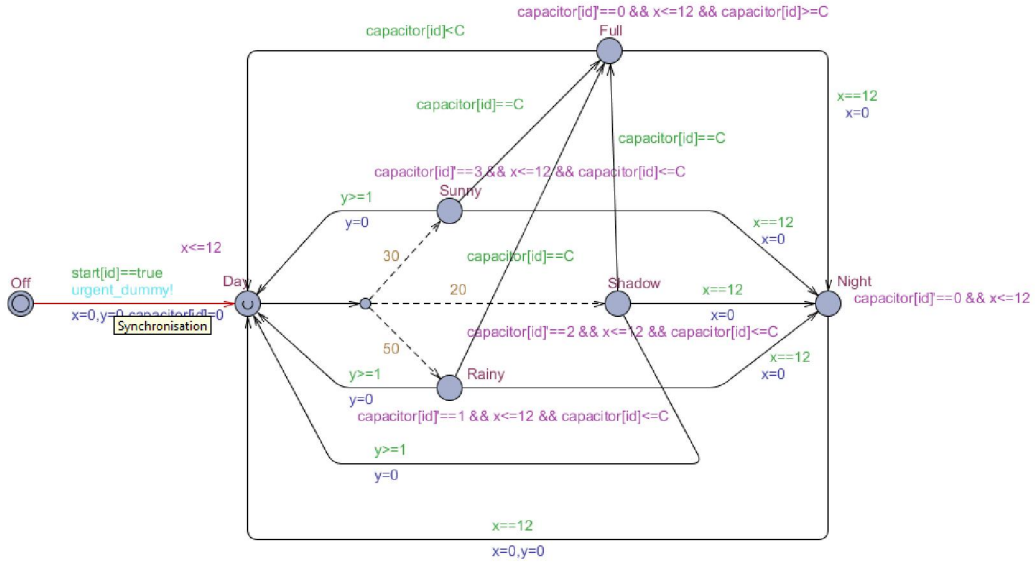
Local declarations for P_EnergyRead

clock x;

const int costRead = 5; //costRead

int M;

Harvester



Local declarations for Harvester

clock x;

clock y;

Dummy

Local declarations for Harvester

None

Global declarations

```
//constants
const int NB = 1;
const int OBS = 2;
const int NUM_NODES = 8;
const int NUM_MAX_RECEIVE = 20;
const int NUM_QUEUE = 6;
const int CHANGE = 200;
const int K = 100;
const int B = 1000;
const int C = 1000;


typedef int[0,NUM_NODES-1] number;
typedef int[0,NUM_MAX_RECEIVE-1] number_receive;

const int ENVIRONMENT[NUM_NODES] = {0,111,222,333,444,555,666,777};

const int SENSE_PERIOD[NUM_NODES] = {0,110,105,87,53,64,42,31};
const int RECEIVE_PERIOD[NUM_NODES] = {10,10,10,10,10,10,10,10};
const int READ_PERIOD[NUM_NODES] = {30,30,30,30,30,30,30,30};

const int PHYSICAL_DIST[NUM_NODES] = {0,500,1000,800,1200,1000,1200,1300};


//global parameters
clock t;
clock capacitor[NUM_NODES];

bool start[NUM_NODES];
bool obsEvent[NUM_NODES];

bool receiveEvent[NUM_NODES][NUM_MAX_RECEIVE];
bool received_NB[NUM_NODES][NUM_NODES];
bool received_OBS[NUM_NODES][NUM_NODES];
bool sense_obs[NUM_NODES][NUM_QUEUE];


bool sending[NUM_NODES];

bool energyEvent[NUM_NODES];
```

```
bool visible[NUM_NODES][NUM_NODES] = {{true,false,false,false,false,false,false,false},
                                      {true,true,true,false,false,false,false,false},
                                      {false,true,true,false,true,false,false,false},
                                      {true,false,false,true,false,true,false,false},
                                      {false,false,true,false,true,false,false,true},
                                      {false,false,false,true,false,true,true,false},
                                      {false,false,false,false,false,true,true,true},
                                      {false,false,false,false,true,false,true,true}};


int next[NUM_NODES] = {0,0,1,0,2,3,5,6};

int NbView[NUM_NODES][NUM_NODES] = {{0,500,1000,800,1200,1000,1200,1300},
                                    {0,500,1000,800,1200,1000,1200,1300},
                                    {0,500,1000,800,1200,1000,1200,1300},
                                    {0,500,1000,800,1200,1000,1200,1300},
                                    {0,500,1000,800,1200,1000,1200,1300},
                                    {0,500,1000,800,1200,1000,1200,1300},
                                    {0,500,1000,800,1200,1000,1200,1300},
                                    {0,500,1000,800,1200,1000,1200,1300}};


int battery[NUM_NODES] = {1000,1000,1000,500,1000,1000,1000,1000};
int battery_tmp[NUM_NODES] = {1000,1000,1000,1000,1000,1000,1000,1000};

int m[NUM_NODES];

int envir[NUM_NODES];
int obs_tmp[NUM_NODES];

int obs_r[NUM_NODES][NUM_MAX_RECEIVE];
int obs_s[NUM_NODES][NUM_QUEUE];
int src[NUM_NODES][NUM_MAX_RECEIVE];

int dst[NUM_NODES];

int send_msg_type[NUM_NODES][NUM_QUEUE];
int receive_msg_type[NUM_NODES][NUM_MAX_RECEIVE];
int send[NUM_NODES][NUM_QUEUE];


int p[NUM_NODES];
int q[NUM_NODES];
```

```
int adjust[NUM_NODES];
int adjust_tmp[NUM_NODES];

int total_msg;
int lost_msg;
int correct_msg;


//channel definitions
urgent broadcast chan urgent_dummy;
broadcast chan senseEnvironment[NUM_NODES];
broadcast chan startReceive[NUM_NODES];
urgent broadcast chan sendMsg[NUM_NODES];

//methods

int n;
void clear(int p) { for(n:number) {received_NB[n][p] = false; received_OBS[n][p] =
false;} }


void updateNext(int p) {int a = 30000; int b = 30000;
                        if(visible[p][0]==true) next[p]=0;
                       else { for(n:number)
                                {if (visible[p][n]==true && NbView[p][n] < a)
                                    {a = NbView[p][n];next[p] = n;} }
                              if(next[p]==p){visible[p][p]=false;
//inconsistent
                                 for(n:number)
                                     {if (visible[p][n]==true && NbView[p][n]
< b)
                                         {b = NbView[p][n];next[p] = n;} }

adjust[p]=K+b-(PHYSICAL_DIST[p]+B-battery[p]);
                                 if(adjust[p]<0) adjust[p]=0;
                                 visible[p][p]=true;

                               } } }

void updateQueue_p(int a) {
                          p[a]++;
                          if(p[a]==NUM_QUEUE) {p[a]=0; }
```

```
                    }

void updateQueue_q(int a) {
                    q[a]++;
                    if(q[a]==NUM_QUEUE) {q[a]=0; }

                    }

int abs(int a, int b) {
                    if(a>=b) return a-b;
                    else return b-a;
                    }
```

System declarations

```
//Node0, the base station
P_Unit0 = P_Unit(0);
//P_Sensor0 = P_Sensor(0);
P_Receiver0 = P_Receiver(0);
//P_EnergyRead0 = P_EnergyRead(0);
//Harvester0 = Harvester(0);
//Sensor0 = Sensor(0);
//Transmitter0 = Transmitter(0);
Receiver0 = Receiver(0);

//Node1
P_Unit1 = P_Unit(1);
P_Sensor1 = P_Sensor(1);
P_Receiver1 = P_Receiver(1);
P_EnergyRead1 = P_EnergyRead(1);
Harvester1 = Harvester(1);
Sensor1 = Sensor(1);
Transmitter1 = Transmitter(1);
Receiver1 = Receiver(1);

//Node2
P_Unit2 = P_Unit(2);
P_Sensor2 = P_Sensor(2);
P_Receiver2 = P_Receiver(2);
P_EnergyRead2 = P_EnergyRead(2);
Harvester2 = Harvester(2);
Sensor2 = Sensor(2);
Transmitter2 = Transmitter(2);
Receiver2 = Receiver(2);

//Node3
P_Unit3 = P_Unit(3);
P_Sensor3 = P_Sensor(3);
P_Receiver3 = P_Receiver(3);
P_EnergyRead3 = P_EnergyRead(3);
Harvester3 = Harvester(3);
Sensor3 = Sensor(3);
Transmitter3 = Transmitter(3);
Receiver3 = Receiver(3);

//Node4
```

```
P_Unit4 = P_Unit(4);
P_Sensor4 = P_Sensor(4);
P_Receiver4 = P_Receiver(4);
P_EnergyRead4 = P_EnergyRead(4);
Harvester4 = Harvester(4);
Sensor4 = Sensor(4);
Transmitter4 = Transmitter(4);
Receiver4 = Receiver(4);

//Node5
P_Unit5 = P_Unit(5);
P_Sensor5 = P_Sensor(5);
P_Receiver5 = P_Receiver(5);
P_EnergyRead5 = P_EnergyRead(5);
Harvester5 = Harvester(5);
Sensor5 = Sensor(5);
Transmitter5 = Transmitter(5);
Receiver5 = Receiver(5);

//Node6
P_Unit6 = P_Unit(6);
P_Sensor6 = P_Sensor(6);
P_Receiver6 = P_Receiver(6);
P_EnergyRead6 = P_EnergyRead(6);
Harvester6 = Harvester(6);
Sensor6 = Sensor(6);
Transmitter6 = Transmitter(6);
Receiver6 = Receiver(6);

//Node7
P_Unit7 = P_Unit(7);
P_Sensor7 = P_Sensor(7);
P_Receiver7 = P_Receiver(7);
P_EnergyRead7 = P_EnergyRead(7);
Harvester7 = Harvester(7);
Sensor7 = Sensor(7);
Transmitter7 = Transmitter(7);
Receiver7 = Receiver(7);

//system.
system P_Unit0,P_Receiver0,Receiver0,
    P_Unit1,P_Sensor1,P_Receiver1,P_EnergyRead1,Sensor1,Transmitter1,Receiver1,Harv
ester1,
    P_Unit2,P_Sensor2,P_Receiver2,P_EnergyRead2,Sensor2,Transmitter2,Receiver2,Harv
```

ester2,
P_Unit3,P_Sensor3,P_Receiver3,P_EnergyRead3,Sensor3,Transmitter3,Receiver3,Harvester3,
P_Unit4,P_Sensor4,P_Receiver4,P_EnergyRead4,Sensor4,Transmitter4,Receiver4,Harvester4,

P_Unit5,P_Sensor5,P_Receiver5,P_EnergyRead5,Sensor5,Transmitter5,Receiver5,Harvester5,

P_Unit6,P_Sensor6,P_Receiver6,P_EnergyRead6,Sensor6,Transmitter6,Receiver6,Harvester6,

P_Unit7,P_Sensor7,P_Receiver7,P_EnergyRead7,Sensor7,Transmitter7,Receiver7,Harvester7,Dummy;