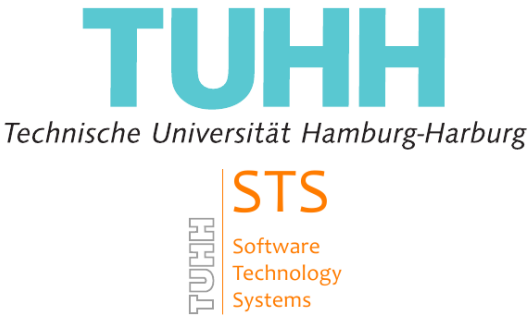

Project Work

Modelling and Verification of QoS properties of a Biomedical Wireless Sensor Network

Author:
Emmanouil Fanourgakis

Supervisor:
Prof. Dr. Sibylle Schupp

February 2012

Abstract

Biomedical Wireless Sensor Networks are small size sensor networks used in medical care applications which transmit vital body information of the patients to a sink node. Meeting QoS requirements such as keeping nodes connected to the sink, or maintaining a high packet delivery ratio and a low packet end-to-end delay, can be life-critical for patients, while minimizing power consumption can be quite important as well. One of the ways to tune such a network, is to change the temporal configuration of the nodes, namely the active and the sleep period duration, in order to save energy against performance. In the existing literature, mostly simulation approaches have been used to analyze sensor networks.

In this paper, we have remodelled an existing Biomedical Sensor Network consisting of nodes that use the Chipcon CC2420 transceiver and follow the IEEE 802.15.4 wireless standard. Like the reference model, we use the Uppaal toolbox and model the wireless nodes as timed automata, i.e. finite state automata enriched with real-time clock variables. We have verified with the use of Uppaal the QoS properties of the network, namely connectivity and packet delivery ratio, and tuned its temporal configuration under various topologies, trying to decrease power consumption and to observe the effects on QoS properties.

The results show that nodes that forward packets to the sink need longer active periods, while the opposite applies to collision points in the network. Our model faces the state space explosion problem, despite the careful design. The reference model reveals a similar behaviour and similar problems, but scales better because of design simplifications. These simplifications are, as we argue, in part unrealistic, yet present the unavoidable compromise, if formal modelling and verification is to be applied to wireless sensor networks of non-trivial size.

Contents

Chapter 1

Introduction

A Wireless Sensor Network is mainly a network of sensors equipped with wireless transceivers. Its task is to monitor environmental conditions within a certain small or larger area and transmit it to central processing points. The importance of such networks is very high due to their innumerable uses, the easiness of installation and the low costs involved. There are however many restrictions that should apply in order that a sensor network functions properly. These restrictions include the ability to operate under very limited resources, namely battery and transmission power and the challenge to adapt to on the fly network topology changes.

A **Biomedical Wireless Sensor Network** contains sensors installed in patients in order to measure vital body functions such as temperature or heart rate, and inform medical staff. Purpose of the network is the efficient and on-time delivery of these measurements, in order to provide feedback for medical staff. The use of Biomedical Sensor Networks introduces high benefits for health care, such as making continuous monitoring of patients possible, and increase the number of patients to be treated effectively in emergency situations.

Because of the nature of their use, ensuring high QoS for Biomedical Sensor Networks can be critical for the life of patients. Relevant QoS properties originate from the requirements for efficient and on-time delivery of the measurements. The nodes should connect directly or indirectly to the sink in regular time intervals, the packet delivery ratio should be adequately high, and the end-to-end delay for the packets should be bounded by a specified value.

To mitigate the issue of limited power resources and increase the lifetime of a Biomedical Sensor Network, the nodes alter their state between active and sleep in regular periods. The longer the sleep periods, the more energy is saved, but also the more the network performance degrades, as sleeping nodes are not able to forward packets of other nodes towards the sink. Thus, for every individual network topology, there exists an optimal compromise point of its *temporal configuration* between energy saving and network performance.

A decent low cost and low power solution is the **Chipcon CC2420 transceiver**[1], very commonly used in wireless sensor networks. Its behaviour can be described by a finite state machine alternating mainly among receive, transmit and idle states.

A **timed automaton** is a finite state machine enriched with real value clocks. Clocks can

be reset to track time between events and state transitions. A timed automaton can effectively describe the behaviour of a real time system, such as a traffic light.

Uppaal is a toolbox for modelling real time systems as networks of timed automata. The user can design templates of automata and encode states, transitions and constraints, in order to represent the behaviour of system units. To create the network, templates are instantiated as nodes. User can query for the satisfaction of network properties using TCTL (Timed Computation Tree Logic). Uppaal also offers a simulation interface, so that user can 'debug' its model by viewing state progression and the value of all network variables and clocks. A known issue of such models is the state space explosion, the exponential growth of the state space to the number of objects in the model. Unless correct precautions are taken during the model design, it is possible that even relatively small systems will take in practice infinite time to be verified.

So far, mostly simulation approaches have been used to analyze the behaviour of wireless sensor networks. Various simulation tools have been developed, some of them being able to handle effectively even huge networks. Despite the problems in scalability, formal modelling is still a valuable tool for the early design phases of a network.

This paper has been inspired by a work of Tschirner et al.[2] that constructs in Uppaal a formal model of a Biomedical Sensor Network, consisting of nodes that use the Chipcon CC2420 transceiver and follow the IEEE 802.15.4 wireless standard. The QoS properties of this model, (*reference model* from now on), are verified against different temporal configuration settings. Finally, the OMNET++ simulator is used to repeat the tests and compare results.

In this work, we have remodelled the Biomedical Sensor Network in Uppaal. We have verified the QoS properties of the network under various topologies and temporal configuration settings, looking for the optimal compromise between energy save and network performance.

The results show that the intermediate nodes that forward packets to the sink need longer active periods. On the other hand, we need to decrease the active time to achieve better performance for specific nodes where collisions occur. The major issue the model faces is the state space explosion problem, which does not allow networks larger than 3 nodes to be verified. We also experience a collision issue due to simultaneous forwarding attempts, because inserting random waits into the model is not possible for state space reasons. The comparison with the reference model shows a similar behaviour and similar problems, but better scaling because of design simplifications. These simplifications, even unrealistic ones, is the unavoidable compromise in order that formal modelling and verification can be applied in non-trivial size networks.

The next sections are organized as follows: **Chapter 2** gives the definition of Biomedical Sensor Networks and their quality properties, as well as describing how temporal configuration can save energy and affect performance. It also gives a short definition of timed automata and introduces the Uppaal platform and the state space explosion issue. Finally, it gives a view in the existing literature concerning sensor networks. **Chapter 3** presents the network topologies examined and the model created for this paper, the design decisions met and the precautions

taken during design to reduce state space. **Chapter 4** analyzes the verification process for the QoS properties as well as the results. **Chapter 5** compares our model with the reference model, analyzing similarities and differences and demonstrating the results from the verification of the same networks. **Chapter 6** summarizes the conclusions and gives directions for future work.

Chapter 2

Foundations

In this chapter, we shall introduce the basic terms used in the paper. We will give the definition of a Biomedical Sensor Network and its importance in medical care, as well as the QoS requirements involved due to its critical task. We will also analyze how energy for its nodes can be saved, with a cost to network performance, by altering the temporal configuration. We will also describe the Chipcon CC2420 transceiver that we will use in the nodes of the network we model. Finally, an introduction on timed automata and a description of the Uppaal toolbox is given.

2.1 Biomedical Sensor Networks

2.1.1 Definition

A wireless sensor network is formed by sensor nodes equipped with wireless transceivers. It covers a small or large geographical area and its task is to monitor environmental conditions like temperature, pressure or pollution, and send the sensed information through wireless transmission to processing spots. Because of the easiness of the installation and the low maintenance costs of such networks, it is possible to monitor hard to access or dangerous locations. This is particularly useful in business and factory automation[3].

In order for a wireless sensor network to function correctly, many restrictions should apply because of limited node resources, namely battery duration and transmission power. Due to the short wireless range, packets may need to be forwarded many times through the network to reach their destination. Additionally, common requirements include flexibility to allow on-the-fly network topology changes and node mobility, unattended operation, and tolerance to individual node failure.

In a Biomedical Sensor Network, sensors installed in patients can measure vital body functions such as temperature, heart rate, and blood pressure[5]. These information is transmitted through the wireless network to a sink node where it can be evaluated and generate feedback for medical staff. Such networks are of small size, containing as many as some tens of nodes[2]. Purpose of the network is the efficient and reliable delivery of the critical information.

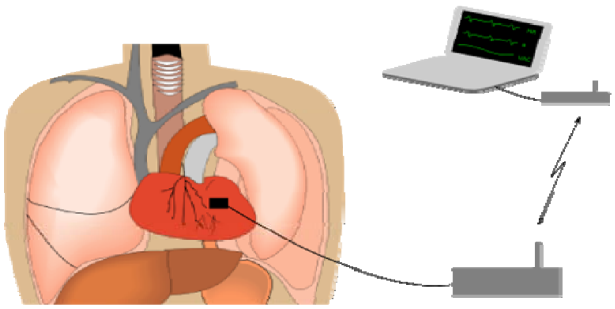

Figure 2.1: A medical sensor wireless network (Source: [4])

By using such systems, continuous monitoring of patients is possible, as well as instant notifications for changes in their health status[5], both at low costs. Not only intensive care units can strongly benefit, but also regular treatment. It is highly important that by using such systems in case of emergency situations like natural disasters, the number of patients that is possible to be treated can substantially grow. Additionally, it is possible to create medical databases with long term patient data for use in medical research[6]. On the other hand, the nature of Biomedical Sensor Networks imposes even more functional constraints and challenges. On-time and efficient delivery of messages can be life-critical for patients.

2.1.2 QoS requirements

Quality of Service is a somehow abstract term. In this work, we will focus on the efficient and on-time delivery of sensor measurements to the sink node

An important requirement is that every node gets to connect with the sink at least once in a specified time frame. This connection will be either directly, because of a direct wireless link, or through packet forwarding. We shall call this requirement **connectivity to the sink**[2].

It is also important that a reasonable amount of the total packets produced in each node is finally delivered to the sink. The **packet delivery ratio** should, thus, be equal or higher than a percentage specified by the requirements.

Finally, it is of equal importance not only that packets are delivered to the sink, but also that they are delivered on time. Therefore, the **end-to-end delay** for the delivery of every packet should be equal or shorter than a reference time specified by the requirements.

2.1.3 Temporal configuration of the nodes in a sensor network

In order to save energy and extend their lifetime, the nodes of a wireless sensor network can alter their state between active and sleep as can be seen in Figure 2.2. An active period will be followed by a sleep period, then a new active period and so on. The node is able to transmit

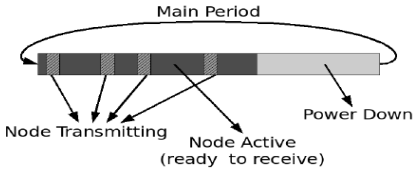

Figure 2.2: Active and sleep periods of a sensor node (Source: [2])

or receive only during the active period. The energy consumption of the sleep state is usually several orders of magnitude smaller than the one of the active mode. By extending the sleep periods, we increase the amount of the energy saved, and thus the node lifetime.

If the sole purpose of the node is to use its sensor to collect measurements and transmit them, then the active period can be reduced to a small fraction of the node life cycle. However, as mentioned before, because of the limited transmission range, packets in a wireless sensor network often have to be forwarded in order to reach the sink. Thus, every node plays also an essential role as an intermediate forwarding point for packets of other nodes. As during its sleep period the node is unable to do any forwarding, packets from other nodes will have fewer chances of reaching the sink. Obviously, depending on the network topology, some nodes will need more active time in order to be able to forward the packets they receive. Therefore, there is a compromise between the energy saving and the ability of the network to deliver packets. The optimal point is different for every specific network according to the available resources and the performance requirements.

2.1.4 The Chipcon CC2420 transceiver

The Chipcon CC2420 transceiver is a very commonly used hardware in sensor networks. It is a single small-size, low-power and low-cost solution for wireless applications suitable for use with low cost micro-controllers. It offers an effective data rate of about 250 kbps and high sensitivity up to -94 dBm. It runs at 2.4 GHz and uses 0.18 μm CMOS technology[1].

The chip behaviour can be understood as a state machine, a simplified diagram of which we can see in Figure 2.3. Initially, the chip is turned off, so a warming up and stabilization time is necessary. After that, the chip is in the idle (power-save) state, from which it can quickly change into the active state awaiting for transmissions and receptions to be done. In the end of a transmission or a reception, the chip goes back to the active state. Reentering the idle state is possible from all states.

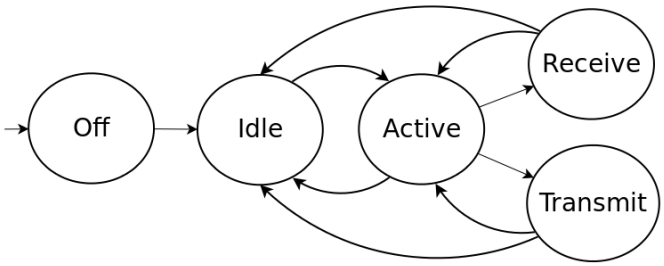

Figure 2.3: State machine of the CC2420 transceiver

2.2 The Uppaal platform

2.2.1 Timed Automata

A timed automaton is a finite state automaton equipped with a set of real-valued clocks[7]. The clocks are *synchronized*, thus they proceed in the same pace as time advances[8]. A clock can be individually reset. In that case it keeps track of the time since the reset.

With the help of these clocks, it is possible to track the time between events in the automaton and code time restrictions in its behaviour. For instance, to put a constraint in a transition, an expression on clocks called *guard*, is assigned to it. The transition cannot be taken if the guard is not satisfied.

Timed automata provide a framework for modelling *real-time systems*, systems for which the correct behaviour is dependent on timing constraints. There are many examples of such systems in everyday life, such as traffic lights.

2.2.2 The Uppaal platform

Uppaal is an integrated environment for modelling systems as networks of timed automata enhanced with integer variables. Uppaal provides the functionality to simulate models and verify their properties. Uppaal is jointly developed in the Universities of Uppsala and Aalborg. Since the first version in 1995, development still goes on[9]. A commercial and an academic version are available. Uppaal uses a client-server architecture with a Java-based graphical user interface and a model checking engine (server) that can run on various operating systems. Command line interface for verification, is also available.

The user will at first design the model in the graphical user interface. Uppaal supports parametric node templates for effective representation of multiple similar objects. *Channels* assist the communication between objects, allowing message passing. States and transitions have to be defined for each template. Clocks and integer variables can be added both as local in a template, or globally. In transitions, clock and variable expressions can be assigned as

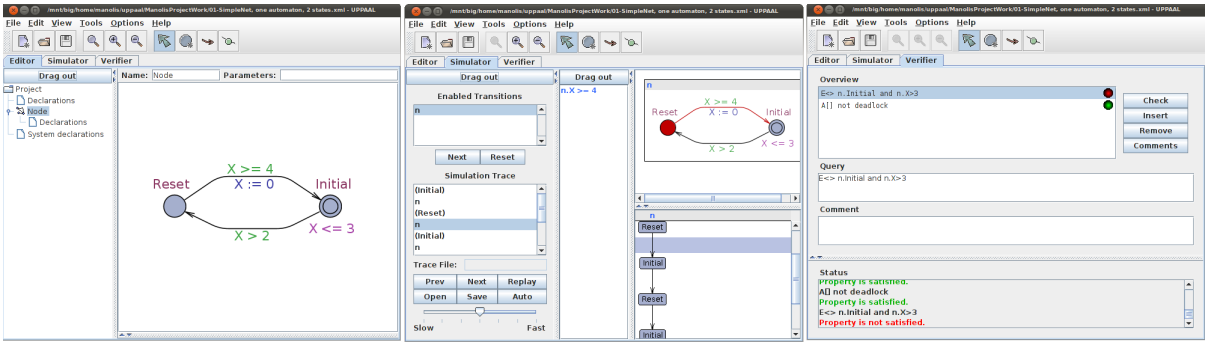

Figure 2.4: The Uppaal Graphical User Interface: Model designer, simulator and verifier

Guards, *Syncs*, and *Updates*, to express constraints to execute the transition, conditions that triggered it, and actions executed by it respectively. In a similar way, *invariants* can be assigned to states in order to express conditions to be met when the node is in the state. Expressions and statements can be grouped in functions using a c-like language for which a simple text editor is available. Finally, in global declarations, nodes of the network are instantiated from the templates.

Uppaal provides the capability to simulate models. Users can run and 'debug' the model having a view of the system state and the contents of all clocks and variables, and being able to choose in every state which transition among the valid ones to execute for going to the next state. It is possible to save the current state of a run as a *system trace* file. From that file, the state can be loaded in order to continue the simulation. There is even an auto play feature, for observing a random run of the model.

For verification, Uppaal provides a query editor. Users can form queries about model properties using a temporal logic language, subset of **TCTL** (Timed Computation Tree Logic). Quantifiers such as A□ and E◊, 'for all possible states along all possible paths...' and 'for at least one state in at least one path...' are available respectively. In the case that the property queried is not satisfied, Uppaal can generate a counterexample as a system trace and load it, so that the user can examine the system status during the counterexample.

Uppaal offers some parameters to be tuned regarding the model checking engine. Users can choose among various algorithms such as breadth-first or depth-first for searching in the model, different levels of state space reduction techniques, and the option to use over-approximation or under-approximation. Batches of queries can be easily verified using the command line interface. Within the limitations of the Uppaal platform, it should be mentioned that no parallel processing is supported.

2.2.3 Reducing state space of models

Because the state space of formal models is exponential to the number of objects, simply adding an object can increase tremendously the number of states to be checked. This issue is known as the **state space explosion problem**. In order to keep the state space small, the desired model complexity should be defined so that the implementation can be as simple as possible, focusing only in the required details. Unless the correct precautions are taken, even systems with relatively low number of objects may not be able to be verified within reasonable time.

The number of variables, clocks and channels to be used shall be kept to the minimum. The integer variables should preferably be declared as bounded, in order to limit the possible values. It is of equal importance to reset variables when their value is not important.

The number of states should also be kept to a minimum and possible state interleaving should be minimized. Uppaal provides a useful state option, the *committed states*. After the model visits a committed state, the transition to one of the successor states should be taken immediately. This means that time does not pass in a committed state. Because of these properties, adding a committed state adds less complexity to the model than adding a normal state.

Another useful structure provided by the Uppaal is the *scalar sets*. A scalar set is an integer type, the elements of which cannot be distinguished. Only assignment and identity testing are available for scalar sets, not ordering and comparison. Scalar sets are particularly effective for representing symmetric objects, thus applying **symmetry reduction** to the model in order to reduce state space.

2.3 Related work

In the literature, mainly simulation-based approaches are used to analyze the behaviour of wireless sensor networks. Various tools have been developed. Avrora[10] can simulate AVR microcontroller-based sensors in a cycle-by-cycle instruction basis, scaling up to thousands of nodes, with the capability to handle simulation of tens of nodes in real time. To measure the performance of a sensor network, the most well-used scheme is through a discrete event simulator such as OMNET++[11], using extensions that accurately describe the behaviour of wireless sensor components. One of them is Castalia[12], written on top of OMNET++.

Formal models, on the other hand, have issues with scalability, because of the state space explosion problem. However, formal modelling can be valuable for the initial design phase applying on small size networks, because it takes into account all possible scenarios, notably the worst case. It can also reveal possible problems in the implementation of the relevant protocols.

Tschirner et al. have created a formal model of a Biomedical Sensor Network using the Uppaal platform. Their network consists of nodes using the Chipcon CC2420 transceiver and follows the IEEE 802.15.4 wireless protocol. They tested the network under various temporal

configurations in order to minimize energy consumption under certain QoS requirements. They also developed an optimized version of the Uppaal simulator that allows them to efficiently run tests and gather statistics in networks up to 16 nodes. They have conducted an OMNET++ simulation as well, and showed that it produces similar results, though OMNET++ describes more effectively the wireless channel[2].

Chapter 3

Modeling the network

In this chapter, we will present our model. At first, we describe the network topologies of two and three nodes that we model. Then, we give a detailed description of the node template in Uppaal, as well as the design decisions and the considerations taken into account to keep the state space small.

3.1 Network Topologies

We choose to model small networks as can be seen in Figure 3.1. There will always be one sink connected to the nodes. In the two nodes network, node 1 acts as an indirect connection between node 2 and the sink. In network 3a, the sink connects to node 1, node 1 connects to node 2 and node 2 connects to node 3. Finally, in network 3b, nodes 2 and 3 are directly connected to the sink, while node 1 is connected to both of them. Networks of greater size were not modelled due to the state space explosion.

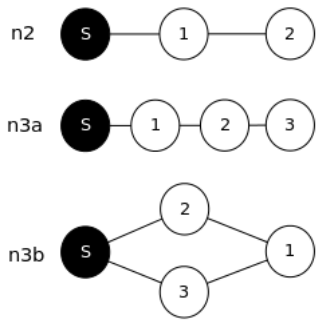

Figure 3.1: Network topologies examined

Among the requirements of the network, is the ability to adapt to topology changes. This is a real life requirement, since the sensor nodes are installed in possibly moving patients. Furthermore, it is realistic to assume that nodes are not aware of the location of the sink. The

only information a node should be aware of, is the existence of the nodes to which it is directly connected. With such a design, we have a highly flexible network, in which we can on the fly relocate, add, or subtract nodes and sinks.

From the previous, it is obvious that for such a design, a fixed routing scheme is not the best choice. To adopt a suitable mechanism in order to give messages a chance to reach the sink, we choose the *controlled flooding* scheme. In this scheme, each node forwards every packet received to all its neighbour nodes, including the sender. Each node remembers the packets it has already received and does not forward them again. This avoids unnecessary flooding in the network.

The use of acknowledgements is also important in order to decrease the chance of messages getting lost along the way to the sink. After node A has transmitted a message to node B, it waits for the acknowledgement. Node A will regard as an acknowledgement the packet retransmission from node B. If no acknowledgement is received before a timeout expires, node A will retransmit.

It is possible that collisions may be encountered if many neighbour nodes transmit at the same time within a node's listening range. Against collisions, the IEEE 802.15.4 wireless protocol specifies a back-off mechanism. Each node can sense the channel only when *not* transmitting. When about to transmit, every node checks whether the channel is active, that is, whether somebody is already transmitting within the node's listening range. If not, the node starts transmitting itself. If, on the other hand, there is already a transmission in the channel, the node will back-off for a random period of time and then retry.

Every node will produce a packet of measurements once in its life cycle. The life cycle of a node, however, is way longer than the packet transmission time. Therefore, a node will spend only a small fraction of its time transmitting. For small size networks, even if the node has to forward packets from other nodes, it is reasonable to assume that the network traffic will be sparse or moderate. In addition to that, during the life cycle of the node, its packet has the chance to be forwarded many times. So, we can assume that the life cycle of the node should be enough time for the packet to reach the sink.

3.2 The sensor node

In the diagram in Figure 3.2, every individual node is identified by its `ID`. All nodes except for the sink have a non-zero ID, the sink has a zero. Each node except for the sink produces packets containing the sensed measurements. These packets are identified by the ID of the generator node. The variable `outg` marks for every node the current packet for transmission. A value of zero means there is no packet to transmit. The global array variable `inc[]` marks the current incoming package per node, the one or ones being transmitted in the node's range. A zero and a negative value denote no active transmissions and a collision respectively. The global double array `f_s[][]` stores for every node the forwarding status for packets generated in all nodes including itself. For example, `f_s[x][y]` is the status for node `x` concerning packets from node

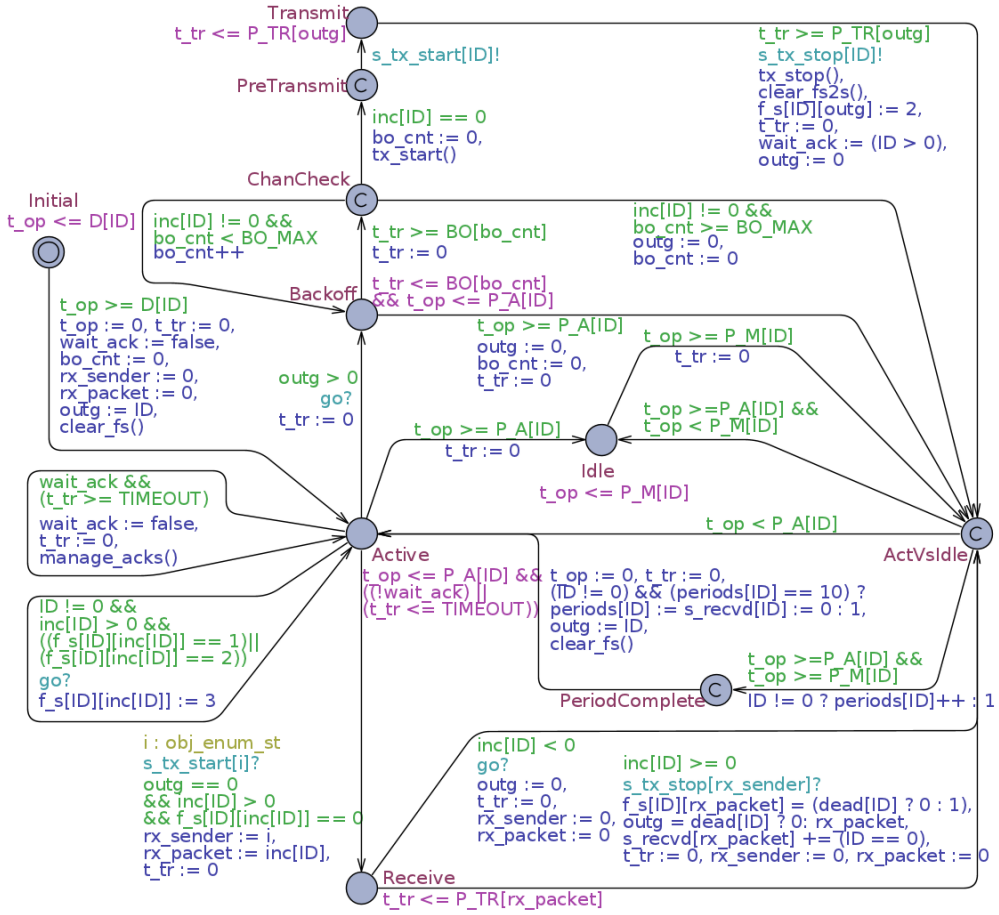

Figure 3.2: Model of a node

y. A value of zero means that the node did not receive the packet yet, 1 and 2 both represent a received and forwarded packet waiting for an acknowledgement, and 3 stands for a forwarded packet with the acknowledgement already received. The variable `bo_cnt` counts the number of back-off waits during transmission. The variables `rx_sender` and `rx_packet` store temporarily the sender and the generator node of the packet during the reception. Finally, `periods[]` and `s_recvd[]` are used in the verification, counting the number of periods and packets received in the sink respectively.

Each node has two internal clocks: `t_op` and `t_tr`. The first one is for timing the operation of the node, identifying whether it should be active or sleeping. The second clock is for timing of receptions, transmissions and back-off waiting. The first clock is reset in the beginning of each period, just after the node wakes up from the sleep, while the second gets reset in the end of every reception, transmission or back-off waiting.

The Finite State Machine encapsulating the node functionality starts from the power off (**Initial**) state. Because not all nodes are fully functional at the same time, we modelled an

initial delay per node (`D[ID]`). Afterwards, the node initializes its state and proceeds to the Active state, producing a new set of measurements provided from its sensor. This set forms a packet to be transmitted.

Two critical time settings that are specific for each node are its main and active period. The first is the node's life-cycle. The second is the subset of the first during which the node can transmit and receive. For the rest of the time within one life cycle period, the node stays in sleep mode, in **Idle** state. At the end of the sleep mode, the node wakes up and produces a new packet for transmission. The **ActVsIdle** is an intermediate decision state that leads to Active or Idle depending on the node operation timer `t_op`. Similarly, the **PeriodComplete** is an intermediate state after ActVsIdle, before going again to Active in the case that node has completed a period.

While not in sleep, the **Active** state is the node's main waiting state. Various events trigger its subsequent transitions. If the active period has ended, the node goes into sleep, in the **Idle** state.

When in Active state, if there is a packet to transmit, if thus `outg[ID]` is non-zero, the node follows the transmitting sequence that starts from the **Backoff** state. From there, the node instantly goes to **ChanCheck** state, to check if the channel is free. If not, then it returns to the Backoff state, now waiting for a random period of time. At the end of this time, it checks the channel again. If the channel is still not free, the maximum value of the random waiting in Backoff state will be twice as big. The procedure goes on until the channel is free or the maximum number of back-offs is reached. In the latter case, the node rejects the packet and goes back to the Active state. During any of the back-offs, if the main period is over, the node goes to sleep rejecting the packet to be transmitted. At any point where the node will find the channel free to transmit, it starts transmitting (**PreTransmit** & **Transmit** states). In the end of the transmission, the node returns either to Active or to Sleep state, depending on the operation timer compared with the active period.

During the Active state, the node also listens for new packets transmitted within its channel range. Packets that the node has already received and forwarded, will be ignored. Otherwise, the node starts reception getting into the **Receive** state. In case there is a collision during the Receive state, the node terminates the reception and rejects the partially received package. If no collision occurs, we end up back in Active state with a new packet to forward or into sleep in case the main period is over.

There are two more functions of the node during the active state: Listening for acknowledgements of forwarded packets and waiting for an acknowledgement timeout. When an acknowledgement is received, the corresponding packet is marked to be ignored in the future. In the case of an acknowledgement timeout, the relevant packet will be directly retransmitted.

The sink is modelled as a special case of a node, that neither produces nor forwards packets. It only collects packets from other nodes and gathers statistics. The sink has always an `ID` equal to zero. Its temporal configuration is such that it does not sleep. For implementation reasons

however, the sink will pass from the Sleep state, instantly returning to the Active state.

3.3 Design decisions

It was mentioned that the node will go to sleep after the end of its active period. An exception to this rule is that the node does not stop a transmission or reception in order to go to sleep mode. This makes sense as a design decision in order to prevent 'important' actions from being interrupted. It does not alter the node behaviour to a large extent, since the duration of transmission and reception are very small relative to the main period. We need, however, a new intermediate state, ActVsIdle, which can be introduced as committed though, to avoid issues from active period constraints violated.

When a node goes to sleep, all information from the active period is erased. Variables are reset for packets already sent, waiting for acknowledgements, and packets pending to transmit. So, in the beginning of the next active period, the node will have a fresh state start. This is done in order to aid solving implementation issues.

Concerning the reception of an acknowledgement for a transmitted package from a sender node, the first bytes of the package forwarded from any neighbour node are regarded from the sender as a valid acknowledgement. The sender stops the reception and marks the packet as acknowledged. The sender will not expect acknowledgements from other neighbour nodes concerning that packet. This design minimizes the model complexity. We assure that at least one neighbour has received the packet and will proceed forwarding it. The scheme can survive possible collisions or packet losses, if at every forwarding stage, at least one neighbour node transmits the packet successfully.

According to our design, the node can transmit more packages while waiting for acknowledgements. This sounds like a good approach, as it allows the node to handle multiple pending packets and to be flexible. Each node will remember the acknowledgement state of the packages it has transmitted. This state is stored in the `f_s[][]` array. However, because there is only one timer for acknowledgements per node, every time a transmission ends, this timer is reset by the node. Upon the timer expiration, among all the pending packets, the check for an acknowledgement will be applied for the packet with the earliest transmission time.

In case the acknowledgement of a packet is lost, the sender will attempt to retransmit the packet until its active period ends. This is a non critical issue. The neighbour nodes will simply ignore new transmissions of the packet. Plus, the timeout is selected large enough so that the retransmissions do not happen so often, in order to increase dramatically traffic and collisions in the node's neighbourhood.

The timeout period has to be set to a reasonable value, that should not be very low because of the previous issue mentioned, but should also not be very high such as to decrease the retransmission frequency, and thus the chances of a packet reaching the sink within the main period of its node. We set the timeout duration to such a value, that every packet has the chance to be transmitted at least 10 times.

3.4 State space issues during model design

In order to keep the state space of the model small, so that networks of more nodes can be verified, a careful select of variables has been used. The global double array `f_s[][]` for instance, that represents for every node the forwarding state of packets coming from all nodes, is allowed to take only three possible values, so its type is declared as an integer, bounded to these values. Another example is in the local per node `wait_ack` variable which is declared as boolean because we are interested in only two possible values.

By looking at the model state diagram in Figure 3.2, we see that clocks and variables are constantly reset when their value is not useful. For example, during both transitions from Active to Idle and from Idle to ActVsIdle, the clock `t_tr` is reset, because the node is not going to receive or transmit any package during Idle state. Without these resets, in two similar cases where the model goes from Active to Idle state with different value in `t_tr`, two different traces need to be generated. The clock `t_tr` is also reset in the end of all receive or transmit sequences. Same holds for the `rx_sender` and `rx_packet` variables that are immediately reset when the transmission is over, as well as the back-off counter `bo_cnt` in the end of the back-off sequence.

We can also see in the model state diagram, that there has been extensive use of committed states. Basically, the only states where time is allowed to pass are the Active, Idle, Backoff, Receive and Transmit. In some cases where states had to be introduced to solve implementation issues, like in PreTransmit or PeriodComplete, a committed state was used, and care was taken to avoid branches.

Scalar sets can unfortunately not be used in the current model. If they were used, it would not be possible to distinguish between nodes. So, neither applying individual per node temporal configuration, nor defining a custom topology would be possible. The network would be fully symmetric.

Concerning design decisions mentioned in the previous section, the decision not to interrupt reception or transmission in order to go to sleep, will certainly enlarge the state space, because it creates new possible scenarios for the system trace. Same holds for the decision to allow the node to handle multiple packets and acknowledgements.

Chapter 4

Verifying the network

In the previous chapter, we described the network topologies to be verified and the details of our model. In this chapter, we will analyze the procedure as well as the results of the verification tests on the QoS properties of the Biomedical Sensor Network

4.1 Hardware used and Uppaal options

The hardware used for the verification, was a series of 6 laboratory computers with a dual core Intel Pentium processors and 8 gigabytes of RAM. For the verifications that demanded more memory, a server of four 8-core AMD Opteron processors sharing 128 GB of RAM was used. It should be noted that for every simulation run, Uppaal utilizes only a single core.

CPU type	**RAM (GB)**	**# of machines**
Intel Pentium @3.33 GHz, dual core	8	6
4x AMD Opteron @800 MHz, 8-core	128	1

Table 4.1: Hardware used for verification

In the Uppaal settings, the Difference Bound Matrices option for the state space representation and the 'aggressive' for state space reduction were used. The Random Depth Search option was used for connectivity and the Breadth First Search for the packet delivery verification. The model checker was accessed from the command line with the assistance of utilities like `screen` (create multiple terminals per session, keep session active after logging out), `time` (program resource usage measure), and `script` (terminal logging).

4.2 Deadlock Absence and Sink Connectivity

The first property to be verified in the model, is the fact that it does not fall into deadlocks. This is relatively easy to accomplish, since in Uppaal there is the built-in state formula `deadlock`,

which holds true for all deadlock states. So, we need to ensure that across all states in all possible paths, this formula is not satisfied. The code for the query is thus `A[] not deadlock`. All tested networks under all temporal configurations were successfully verified for the absence of deadlock.

Next, we will verify that each node stays connected with the sink. This will be achieved through the `periods[]` array, which is indexed by the IDs of the nodes, such as each array entry corresponds to a node. For every node, its value in this array is increased upon the completion of a main period. The sink, however, clears the `periods[]` for the creator of every packet it receives. Thus, the `periods[X]` of a node `X` will indicate the number of consecutive packets that did *not* reach the sink.

To ensure that for a node `X` it never happens that `Y` consecutive packets do not reach the sink, we need to verify that across all states in all possible paths, the `periods[X]<Y` property is satisfied. In other words, we require that the node `X` gets connected to the sink at least every `Y` periods. The relevant query is `A[] periods[X]<Y`.

The strategy for running this verification is to start querying for small values of `Y` for each node and increase that value until we get a property satisfaction. A small value of `Y` denotes a strict requirement, as it restricts the nodes to have more often connections to the sink. The smaller the value for `Y`, the easier it will be for Uppaal to find a counterexample and return non satisfaction. We will assist the search for a counterexample by using the *Random Depth First* option for the search order in the Uppaal settings. As `Y` grows larger, fewer counterexamples exist. More states should be checked to find one, so the response will need more resources. At some point, where for some `Y` the property will hold, Uppaal will need to check the entire state space, and will be unable to find a counterexample, returning a satisfaction.

Verification has been run from the command line with the help of a shell script. The "Reuse existing state space" option has been enabled, so as `Y` grows larger, the states already examined for its previous values are retained in memory. The verification of the connectivity did run together with the verification for the absence of deadlock. A synopsis can be seen in Table 5.1.

The networks mentioned in the table are the ones shown in Figure 3.1. We start by giving to every node an active time equal to two thirds of its main period. For every network, there is a row in bold, which marks the optimal configuration, the best compromise between energy saving and good performance.

We can see that for the two-node network, the performance is very good and does not degrade significantly until we decrease the percentage of the active period to 40% for both nodes. On the other hand, in the network 3a, we need to increase the fraction of active time for nodes 1 and 2, from which traffic towards node 3 is routed. Finally, in network 3b, where nodes 2 and 3 act as intermediates between node 1 and the sink, the performance of node 1 is very bad and can not improve significantly by tuning. The best approach is to increase the active time for node 1 and to decrease slightly for the other two nodes.

By observing Table 5.1, we see that the higher the network performance, the smaller the

Network	Temporal conf. (% act./total per node)			Connectivity (periods per node)			States	CPU time	RAM
	1	**2**	**3**	**1**	**2**	**3**			
2	67	67	n/a	2	2	n/a	46 K	5 sec	42 MB
2	50	50	n/a	2	2	n/a	51 K	6 sec	44 MB
2	**40**	**40**	n/a	**2**	**3**	n/a	46 K	4 sec	42 MB
2	40	30	n/a	2	4	n/a	48 K	5 sec	42 MB
2	30	40	n/a	2	4	n/a	48 K	5 sec	42 MB
3a	67	67	67	3	5	4	16 M	1.2 days	73 GB
3a	90	80	80	2	2	3	690 K	143 sec	321 MB
3a	**90**	**80**	**67**	**2**	**2**	**3**	680 K	128 sec	315 MB
3a	80	80	67	2	4	3	3.5 M	24 min	1.4 GB
3b	67	67	67	8	4	4	258 M	3.8 days	62 GB
3b	90	67	67	8	4	4	478 M	6.4 days	62 GB
3b	**90**	**50**	**50**	**8**	**3**	**4**	257 M	3.8 days	63 GB

Table 4.2: Connectivity verification

state space. For network 3a, the number of states in the optimal 90-80-67 configuration is 20 times smaller than in the initial 67-67-67. The explanation for this is relevant to the retransmissions. In the optimal temporal configuration, we have far less retransmissions, and thus simpler scenarios with less possible variations.

4.3 Packet delivery ratio

To verify that the packet delivery ratio stays above a threshold, we will need to modify slightly the model. The introduced `s_recvd[]` array which, like `periods[]`, is indexed by the IDs of the nodes, serves the purpose to count the number of packets per node that are delivered successfully to the sink. To calculate the ratio per node of packets delivered to the sink over the total packets produced, we simply need to divide the `s_recvd[X]` by `periods[X]`, where `X` is the node.

We can clear the value of the `periods[]` array for each node, when it reaches an arbitrary value such as 10. The corresponding value of the `s_recvd[]` array, denoting the number of packets the sink has received from the node, can also be cleared at the same time. So, in the end of every tenth period for every node, the contents of `s_recvd[]` will number the packets that reached the sink.

We can use the `imply` keyword of the Uppaal verifier in order to assure that, for instance, when `periods[1]` is 10, `s_recvd[1]` is bigger than a number, say 7. If we verify this formula across all paths and all possible states, then we know that the packet delivery ratio for node 1 is above or equal to 7/10. The formal syntax of the formula is `A[] periods[1]==10 imply s_recvd[1]>=7`. By using this technique, we can approximate the actual value for the packet delivery ratio. It should be noted that using this technique, we obtain more information than

just a probability. A probability 70% for a packet delivery, would simply mean that it is possible -though unlikely- that an arbitrarily long number of consecutive packets are not delivered. In our case, we can guarantee that 7 out of *every* 10 consecutive packets will be delivered.

The verification strategy here is to start by issuing a query for packet delivery of the highest possible ratio, 100%, and keep querying decreasing this ratio until we get a satisfaction. Like in the connectivity verification, as ratio drops, requirements are less strict, less counterexamples exist, so more states have to be examined. Until at some point the property holds, in which case the entire state space will be examined. Results can be seen in Table 5.2. The verification was unable to complete in many cases, mainly in network 3b, due to the high memory resources required. The numbers stated here should be correct though, judging by the resource escalation. In other words, in all non-finished cases, the verification running, seemed to be examining the entire state space.

Network	**Temporal conf.** (% act./total per node)			**Packet delivery** (% ratio per node)			**States**	**CPU time**	**RAM**
	1	**2**	**3**	**1**	**2**	**3**			
2	67	67	n/a	100	100	n/a	456 K	29 sec	128 MB
2	50	50	n/a	100	100	n/a	314 K	33 sec	138 MB
2	**40**	**40**	n/a	**100**	**80**	n/a	461K	27 sec	121 MB
2	40	30	n/a	100	50	n/a	400 K	17 sec	110 MB
2	30	40	n/a	90	60	n/a	480 K	21 sec	130 MB
3a	67	67	67	90	50	40	318 M	2.5 days	
3a	90	80	80	100	100	80	8 M	12 min	2 GB
3a	**90**	**80**	**67**	**100**	**100**	**80**	7.9M	12 min	1.9 GB
3a	80	80	67	100	70	80	12.9 M		
3b	67	67	67	30	60[1]	70[1]			
3b	**90**	**67**	**67**	**40**[1]	**70**[1]	**80**[1]			
3b	90	50	50	30[1]	80[1]	90[1]			

Table 4.3: Packet delivery ratio verification

By observing the results, we see that, as we tune the temporal configuration, the packet delivery follows the same pattern with the connectivity: very good performance for the two-node network, so we can decrease the active fraction for both nodes up to 40%; a need for an increase in nodes 1 and 2, especially 1 for the network 3a; and bad performance for node 1 in the network 3b.

Looking for the optimal temporal configuration in network 3b, we see that there is a conflict between the connectivity and the packet delivery results. As far as connectivity is concerned, the best configuration is 90-50-50 which has slightly better performance for node 2: connectivity every three instead of four periods. However, as far as packet delivery is concerned, the best configuration seems to be 90-67-67 which improves the weak point of the network, the delivery

[1]Incomplete verification

ratio in node 1 from 30% to 40%, even though it degrades slightly the performance in nodes 2 and 3.

To explain the bad performance in network 3b, let us examine the network topology as seen in Figure 3.1. The sink is directly connected to nodes 2 and 3. However, packets from node 1 need to be forwarded through nodes 2 and 3 to reach the sink. The problem is identified in the collisions at the sink from nodes 2 and 3 trying to forward simultaneously. Because of the model implementation, when a packet is transmitted from node 1, nodes 2 and 3 will receive it and proceed to forward it at the same time. The collision occurs in the sink. Nodes 2 and 3 cannot tell there is a collision, only the sink can. They both wait for an acknowledgement. When the acknowledgement timeout expires at the same time for both of them, they will both try to retransmit the packet, resulting to a new collision. The collision sequence continues until one of the two nodes gets to sleep. This also provides an explanation for the increase in the packet delivery ratio, when decreasing the active time for nodes 2 and 3.

To resolve this issue, a random delay needs to be introduced in the node just before the transmission of a new packet. Thus, after a few transmission attempts, it should happen that only one node is transmitting, the other node should be waiting. However, this modification deteriorates the state explosion problem, making the model impossible to verify.

By comparing the number of states in the same configurations between in the model for connectivity and for deadlock verification, we can see that the second has 10 to 20 times more states than the first one. The introduction of `s_recvd[]` is responsible for this, along with the modifications in the usage of `periods[]`, namely resetting once every 10 periods, not when a packet reaches the sink.

4.4 The state space explosion

As it is already explained, it was not possible to verify a network with 4 nodes. The current model does not scale well.

Network	# of states	CPU time	RAM
3a	320	20600	1740
3b	5600	65600	1480

Table 4.4: Normalized state space and verification resources used (1 ≡ Network 2)

Let us try to compare the state space and verification resources for all networks. Taking the connectivity verification with the initial temporal configuration and normalizing with respect to the smallest figures, those of network 2, we end up in Table 4.4. The state space in network 3a is 320 times larger than network 2, but the one in 3b is 5600 times larger! Obviously the explanation for this is the collisions in network 3b because of the simultaneous forwarding

attempts of nodes 2 and 3. The effect on the state space due to the constant retransmission attempts is tremendous. We also observe that the verification time for network 3b is only 3 times longer than the one for 3a, while the memory consumed is slightly less in 3b. These numbers are probably due to Uppaal optimization techniques.

Chapter 5

Comparison with the reference model

In the previous chapters, we introduced our model and discussed the procedure and the results of the QoS verification for the Biomedical Sensor Network. In this chapter, we shall present the reference model, analyze the similarities and differences with our model, and apply the same verifications in order to be able to compare the results.

5.1 The reference model

The timed automaton for the Chipcon CC2420 transceiver node can be seen in Figure 5.1. The state machine begins with an initial delay in the state `Initial_delay`. A new packet of measurements coming from the sensor is produced, and the node goes to the `RX_SFD_SEARCH` state, its main waiting state. The node will enter the `PowerDown` state when its operational clock `x` shows that the time of the work period (`P_W`) is over. From the `PowerDown` state, the node will come back to the `RX_SFD_SEARCH` state when the time of the main period (`P_M`) is over, starting a new main period and assembling a new package for transmission.

From the `RX_SFD_SEARCH` state, when a packet to be transmitted exists, the node follows the transmitting sequence of the states `Backoff`, `PreTX`, `TX_CALIBRATE`, `TX_PREAMBLE` and `TX_FRAME`. In the two first states, the node will wait until the channel within its listening range is free from other transmissions. In the two last states, the preamble and the main part of the packet are transmitted. In the end of the transmission, the node either returns to the `RX_SFD_SEARCH` state, or to the `PowerDown` depending on the value of its operational clock.

A packet reception occurs when the node is in the `RX_SFD_SEARCH` state without a packet to transmit itself, as soon as another node starts a transmission. As long as the sender node is in a listening range, a check performed in the `PreRX` state, the node goes further to the `RX_FRAME` state, receiving the packet. If a collision occurs during `RX_FRAME` state, the node will discard the packet and go back to `RX_SFD_SEARCH`. Otherwise, the reception ends successfully, and the node goes back to the `RX_SFD_SEARCH` state, with a new packet to forward. During the `RX_SFD_SEARCH`

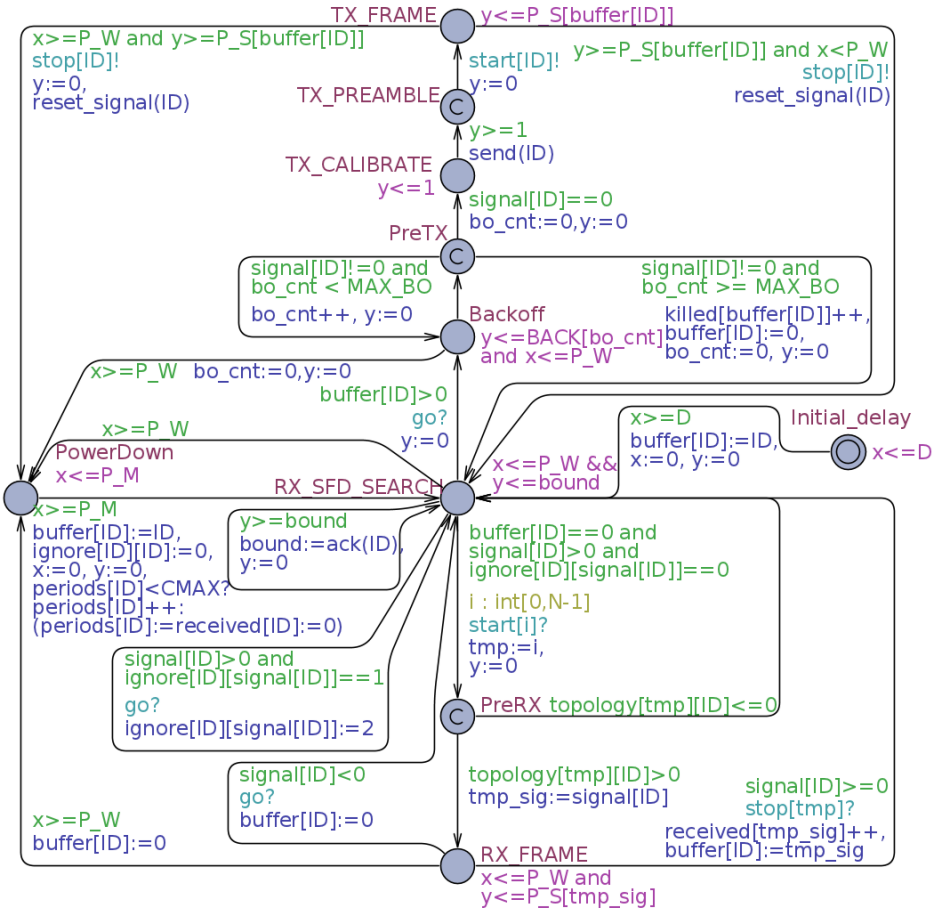

Figure 5.1: The node template in the reference model

state, the node also receives and processes acknowledgements, as well as marking packets for retransmission in case a timeout expires.

We can see in Figure 5.2 the automaton for the sink node in the reference model. The sink starts in the `RX_SFD_SEARCH` state, ready to receive packets. Whenever a node in the network starts transmitting, the sink goes to the `PreRX` state. If the transmitting node is within the sink's listening range, the sink advances to the `RX_FRAME` state for the packet reception. If a collision occurs during `RX_FRAME`, the reception is cancelled. Otherwise, after a successful reception, the sink returns to the `RX_SFD_SEARCH` updating its statistics and setting the packet acknowledgement status to true in all nodes in network.

5.2 Model comparison

By comparing the reference model with the one presented in Section 3.2, we observe that, according to the reference model, when the work period of a node is over during receiving, the node stops the reception immediately, in order to go to `PowerDown` state. In our model, the node continues until the reception is over, and then goes to sleep. The approach of the reference model is simpler and makes more sense. There is no point finishing a packet reception and then ignore the packet to go to sleep.

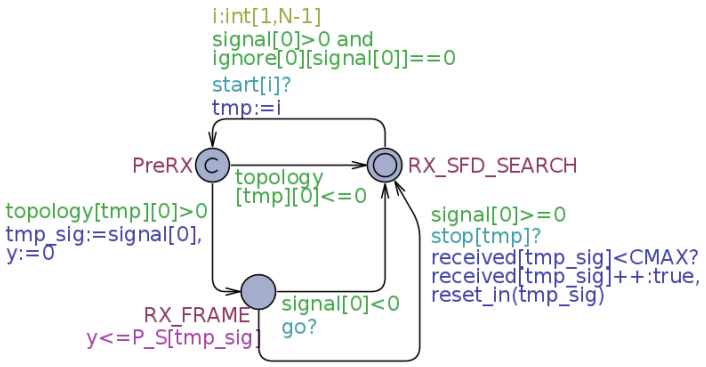

Figure 5.2: The sink template in the reference model

Other than that, the basic structure of the two models is pretty similar, which of course is not a surprise. Both models follow the state machine given for the Chipcon CC2420 transceiver by its manufacturer.

There exists a difference in the acknowledgement behaviour of the sink node not visible in the previous diagram of the node automaton. In our model, the sink produces an acknowledgement for every packet it receives, by transmitting it back to all its connected nodes. In the reference model, the sink node, upon receiving a packet, sets the acknowledgement status to true for this packet in all nodes in the network. This approach is a simplification that certainly prevents the nodes from useless retransmissions of the packet and thus simplifies the model. In a real network, however, the sink node will not have such solutions available.

Another difference not visible in the diagram is that, according to the reference model, the node retransmits a packet only once after a timeout. If the second transmission is also unsuccessful and a new timeout occurs, the node simply marks the packet to be ignored. This is a clever approach to reduce the complexity of the model, as less available retransmissions decrease the total possible number of states. The approach can be realistic. If the two first transmissions fail, chances are that there are collisions in the network, so it is better not to further increase traffic by retransmitting. Alternatively, the packet may be delivered correctly, but the acknowledgement may be lost for any reason. However, in the case that the packet has not been received from intermediate nodes because they are in sleep mode, the best approach is indeed a later retransmission.

5.3 Comparison of verification results

To make the comparison more interesting, we have used the reference model to verify the same network topologies, so that we can compare results for both models.

The reference model was successfully verified for the absence of deadlock in all network topologies. The results for the connectivity verification can be seen in Table 5.1. The first observation is that the reference model scales much better. All topologies can be verified within very short time. This should not be a surprise. Because of the lower complexity and the

Network	Temporal conf. (% act./total per node)			Connectivity (periods per node)			States	CPU time	RAM
	1	**2**	**3**	**1**	**2**	**3**			
2	67	67	n/a	2	2	n/a	650	0.06 sec	22 MB
2	50	50	n/a	2	2	n/a	750	0.06 sec	22 MB
2	**40**	**40**	n/a	**2**	**3**	n/a	790	0.06 sec	22 MB
2	40	30	n/a	2	4	n/a	760	0.06 sec	22 MB
2	30	40	n/a	2	4	n/a	950	0.08 sec	22 MB
3a	67	67	67	3	4	4	4.2 K	0.44 sec	25 MB
3a	90	80	80	3	2	3	4.3 K	0.5 sec	25 MB
3a	**90**	**80**	**67**	**3**	**2**	**3**	3.9K	0.5 sec	25 MB
3a	80	80	67	3	3	3	3.9 K	0.4 sec	25 MB
3b	67	67	67	6	4	4	1.6 M	2 min	350 MB
3b	90	67	67	6	4	5	4.8 M	5.5 min	850 MB
3b	**90**	**50**	**50**	**6**	**3**	**5**	1.2 M	1.5 min	235 GB

Table 5.1: Connectivity verification in reference model

simplifications in design, the state space is indeed smaller.

In addition to that, the sink connectivity property follows here the previously encountered pattern concerning its dependency on the temporal configuration. In networks 2 and 3a, where there is no issue of collisions, the nodes get to connect more often to the sink as the active time of the nodes increases.

Network	Temporal conf. (% act./total per node)			Packet delivery (% ratio per node)			States	CPU time	RAM
	1	**2**	**3**	**1**	**2**	**3**			
2	67	67	n/a	100	100	n/a	3.2 K	0.05 sec	21 MB
2	50	50	n/a	100	100	n/a	3.6 K	0.06 sec	21 MB
2	**40**	**40**	n/a	**100**	**90**	n/a	3.4 K	0.06 sec	21 MB
2	40	30	n/a	100	80	n/a	3.6 K	0.06 sec	21 MB
2	30	40	n/a	100	60	n/a	3.9 K	0.05 sec	21 MB
3a	67	67	67	100	80	80	25 K	0.5 sec	42 MB
3a	90	80	80	100	100	100	27 K	0.5 sec	45 MB
3a	90	80	67	100	100	100	26 K	0.5 sec	44 MB
3a	**80**	**80**	**67**	**100**	**90**	**100**	26 K	0.5 sec	43 MB
3b	67	67	67	40	70	70	1.3 M	21 sec	680 MB
3b	**90**	**67**	**67**	**40**	**70**	**50**	3.4 M	52 sec	1.5 GB
3b	90	50	50	50	80	60	1.2 M	20 sec	500 MB

Table 5.2: Packet delivery ratio verification in reference model

We can also see that the collision issue concerning node 1 in network 3b is also present in the reference model. According to this model too, each node forwards instantly any received packages. Thus, nodes 2 and 3 will attempt to forward simultaneously the packets received

from node 1. To mitigate the problem, we can decrease the active time of nodes 2 and 3 to reduce collisions, as we did in our model.

The results from the packet delivery verification of the reference model can be seen in Table 5.2. Conclusions are similar to the ones from connectivity verification.

Network	**Temporal conf.** act./total (% per node)			**Model in current paper**				**Model in reference paper**			
				Connectivity (periods per node)			**States**	**Connectivity** (periods per node)			**States**
	1	**2**	**3**	**1**	**2**	**3**		**1**	**2**	**3**	
2	67	67	n/a	2	2	n/a	46 K	2	2	n/a	650
2	50	50	n/a	2	2	n/a	51 K	2	2	n/a	750
2	40	40	n/a	2	3	n/a	46 K	2	3	n/a	790
2	40	30	n/a	2	4	n/a	40 K	2	2	n/a	760
2	30	40	n/a	2	4	n/a	48 K	2	4	n/a	950
3a	67	67	67	3	5	4	16 M	3	4	4	4.2 K
3a	90	80	80	2	2	3	690 K	3	2	3	4.3 K
3a	90	80	67	2	2	3	680 K	3	2	3	3.9 K
3a	80	80	67	2	4	3	3.5 M	3	3	3	3.9 K
3b	67	67	67	8	4	4	258 M	6	4	4	1.6 M
3b	90	67	67	8	4	4	478 M	6	4	5	4.8 M
3b	90	50	50	8	3	4	257 M	6	3	5	1.2 M

Table 5.3: Connectivity verification in both models

The performance of both models under the same topologies can be compared in Tables 5.3 and 5.4. In some cases, the reference model has slightly better performance. In the rest of the cases, the performance is exactly the same. The reference model proves to handle more effectively the collision situations in networks 3a and 3b.

It should also be mentioned that the reference model behaves much better, as far as its state space is concerned, when the conditions in the network deteriorate because of collisions or a decrease of temporal resources. This makes sense. When network conditions deteriorate and packets are lost, the nodes will have to retransmit. In our model, constant retransmissions increase hugely the state space. In the reference model this does not happen because of the limited number of retransmissions.

By comparing the state space sizes between networks, we can see that network 3b has a state space three orders of magnitude larger in relation to network 3a in both models. This reflects the complexity created by the collisions in network topology 3b.

Network	**Temporal conf.** act./total (% per node)			**Model in current paper**				**Model in reference paper**			
				Packet delivery (% ratio per node)			**States**	**Packet delivery** (% ratio per node)			**States**
	1	**2**	**3**	**1**	**2**	**3**		**1**	**2**	**3**	
2	67	67	n/a	100	100	n/a	456 K	100	100	n/a	3.2 K
2	50	50	n/a	100	100	n/a	314 K	100	100	n/a	3.6 K
2	40	40	n/a	100	80	n/a	461 K	100	90	n/a	3.4 K
2	40	30	n/a	100	50	n/a	400 K	100	80	n/a	3.1 K
2	30	40	n/a	90	60	n/a	480 K	100	60	n/a	3.9 K
3a	67	67	67	90	50	40	318 M	100	80	80	25 K
3a	90	80	80	100	100	80	8 M	100	100	100	27 K
3a	90	80	67	100	100	80	7.9 M	100	100	100	26 K
3a	80	80	67	100	70	80	12.9 M	100	100	100	26 K
3b	67	67	67	30	60	70		40	70	70	1.3 M
3b	90	67	67	40	70	80		40	70	50	3.4 M
3b	90	50	50	30	80	90		50	80	60	1.2 M

Table 5.4: Packet delivery ratio verification in both models

Chapter 6

Conclusions & Future Work

Our test results show that the intermediate nodes that forward packets to the sink need –not surprisingly– longer active periods. Reducing the active time can, on the other hand, increase performance in points of the network where collisions occur. Because it is not possible to include random waits in the model, an issue of collisions due to simultaneous forwarding attempts exists, which would not be present in a real-life scenario.

The state space explosion appears to be a big issue for our model. We are able to verify networks of only 2 and 3 nodes apart from the sink. The basic guidelines for keeping the state space small were taken into account during the design, namely the sparing use of clocks and variables, bounded if possible, their reset when their value is not useful and the use of committed states. However, the model proves itself to be quite complex. In addition to that, symmetry reduction cannot be applied to reduce state space, because the networks we model are not symmetric.

Comparing our model with the reference model, we observe a similarity in the results. The network faces even exactly the same simultaneous forwarding collision issue. The difference is mainly that the reference model scales much better, because of simplifications in the design. Such simplifications include the restriction to only one retransmission, and the unrealistic automatic acknowledgement in the entire network for packets in the sink.

Limitations and differences of our verification method compared to traditional simulation techniques are apparent. It is easy to guarantee for the best or worst case for a property value, but it is hard to calculate percentages, or *distributions*.

Uppaal is proved to be a sophisticated, flexible, and reliable tool able to handle very big models of hundreds of millions of states that size up to many tenths of gigabytes in memory. The support for multiple processors is, however, a feature we miss, because the multiple core architecture of our workstations and servers can not be fully utilized.

Future work could include testing the system reaction towards changes in the topology network, introducing for example a new node, or modifying the connection scheme during operation. Or calculating the packet end-to-end delay from the generator node to the sink. The packets are created in the beginning of the node's period when the `t_op` clock of the node is

reset. The value of the same clock at the moment the packets reach the sink is the end-to-end delay. We can form queries for the Uppaal verifier in order to verify the worst case value for the end-to-end delay. Packets not reaching the sink will not be taken into account.

Bibliography

[1] Chipcon corporation, ZigBee-ready RF Transceiver, CC24240 data sheet, June 2004.

[2] S. Tschirner, L. Xuedong, and W. Yi. Model-Based Validation of QoS Properties of Biomedical Sensor Networks. In *8th ACM International Conference on Embedded Software*, 2008.

[3] L.Q. Zhuang, K.M. Goh and J.B. Zhang. The wireless sensor networks for factory automation: Issues and challenges. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (2007)*, pages 141 –148, Sept. 2007.

[4] E. Naess-Ulseth. Biomedical Wireless Sensor Network - BWSN. Nordic Innovation Center, Oslo, Norway, 2007.

[5] V. Shnayder, B. Chen, K. Lorincz, T. Jones, T. R. F. Fulford and M. Welsh. Sensor networks for medical care. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, pages 314–314. ACM, 2005.

[6] M. Welsh, D. Malan and B. Duncan. Wireless sensor networks for emergency medical care. *GE Global Research Conference Harvard University*, 11(1):13–15, 2004.

[7] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, April 1994.

[8] S. Yovine. Model checking timed automata. In *European Educational Forum: School on Embedded Systems*, pages 114–152. Springer-Verlag, 1998.

[9] G. Behrmann, A. David and K. G. Larsen. A Tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems*, pages 200–237. Springer, 3185 edition, 2004.

[10] B. L. Titzer et al. Avrora: Scalable Sensor Network Simulation with Precise Timing. In *Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 477–482, 2005.

[11] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference ESM 2001*, pages 319–324.

[12] H. N. Pham, D. Pediaditakis and A. Boulis. From Simulation to Real Deployments in WSN and back. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2007*, pages 1–6, june 2007.