A Dual-Mode Adaptive MAC Protocol for Process Control in Industrial Wireless Sensor Networks

A. Ajith Kumar S.

A Dual-Mode Adaptive MAC Protocol for Process Control in Industrial Wireless Sensor Networks

Doctoral Dissertation for the Degree *Philosophiae Doctor (PhD)* at the Faculty of Engineering and Science, Specialisation in Information and Communication Technology

University of Agder
Faculty of Engineering and Science
2017

Doctoral Dissertation at the University of Agder 149
ISSN: 1504-9272
ISBN: 978-82-7117-843-7

Printed by Wittusen & Jensen
Oslo

Preface

This thesis is a result of a PhD project undertaken as a collaborative effort between the department of computing and the department of electrical engineering at Bergen University College, Norway. The PhD position is registered at the Faculty of Engineering and Science, University of Agder, Norway. The PhD project was funded via the ICT Engineering research programme at Bergen University College.

The research undertaken in this thesis concentrates on the domain of Industrial Wireless Sensor Networks. In the course of the PhD project, several papers have been published in peer-reviewed workshops and conference proceedings presenting the research contributions that have been achieved.

The thesis is organised into two main parts. The first part contains an overview of the work done, summarising the results obtained and putting them into the context of related work. The second part consist of a collection of already published papers.

Acknowledgements

An integral part of a doctoral dissertation is the acknowledgement of the fact that it is a collective process. A process that is influenced by multiple people in separate phases in diverse ways. Here, I would like to acknowledge their part by thanking them for their time, effort and support. Firstly, I would like to thank my supervisors Lars Michael Kristensen, Knut Øvsthus, and Rune Werner Fensli for their invaluable support and guidance throughout the doctoral study. I am particularly thankful to the diverse guidance obtained from them due to their diverse backgrounds. Each of them brought out their own strengths, resulting in an overall development. I appreciate the freedom I had throughout my dissertation in terms of the research path taken, and work environment. Support was always there for any aspirations and plan I had. Whenever I needed any hardware or other equipments, they were readily available. Their door has always been open for me. Our discussions have always been interesting with constant support, realization of boundaries, and separate perspectives.

I also would like to thank Bergen University College for the position provided, and the University of Agder for the registration as a doctoral student. I am thankful to the staff at the department of electrical engineering and the department of computing for their support. In particular, I would like to mention Per Eilif Thorvaldsen for his support. He always made sure I was doing well and had everything I needed. Also, Carsten Helgesen from the computing department, I always felt like I was a part of both the departments equally. All my administrative requirements were smoothly taken care of by them. I would like to extend special thanks to Emil Cimpan, who has helped me understand the requirements that led to the idea of the central protocol in this thesis. Also, to Farzan Jouleh for the hardware support.

During my doctoral study, I had one project linked directly to my thesis with a fellow student at the time and now a PhD, Kent Inge Fagerland Simonsen. I am grateful to him for the work done together, the support provided on site and online, distantly at times. I would also thank my fellow PhD students Florian Mantz and Xiaoliang Wang who convinced me this is a position I would like to take also for the work environment apart from the research interest in itself. This included a trip to Fløyen when I was in Bergen for the interview, which further was extended with many fun trips including ski trips. I also would like to thank my fellow PhD students, Piotr Kazmierczak, and Erik Eikland who together with Florian Mantz, Xiaoliang Wang, and Kent Inge Fagerland Simonsen formed a fun group in the PhD room in the old building. The old building since has been a benchmark to compare with all facilities provided in the new building, and the time we had previously. I would like to thank them for making me feel welcome in a new city and helping me integrate into it. The mentions in the PhD room are incomplete without the newly appointed PhD students who I have spent the time since the old PhD students had completed, particularly, Fernando Macias and Rui Wang. Also, Roberto who has been visiting Bergen University College last two summers.

The year of 2014, has been special in terms of the work done and also about the Forsker Grandprix that I took part in. This has helped me train in skills that would otherwise not be a direct part of the doctoral study. I had the opportunity to get into the competition and obtain guidance in presentation skills. This gave me a unique experience in putting forward my work in simple terms. I would like to thank Silje Vik Pedersen, Marthe Berg-Olsen, my supervisors and all the staff part of the Forsker Grandprix for this opportunity and the training provided. It has been an experience of a lifetime. I would also like to thank my dear friends; Linnea, Nils, Silje, Elizabeth, Gjermund, my department head Per Eilif Thorvaldsen, my supervisors Lars Michael Kristensen and Knut Øvsthus for their presence and support.

The doctoral study in the new city of Bergen would have been hard if not for the support of the people in my personal life. My Eindhoven gang as I would like to say have always been including and supportive throughout the period. Initially, I had many trips to Eindhoven to cope with stay change alone for the first time through my life. I had not felt for quite a while that I had moved from Eindhoven. It is still my most visited destination and my go to place for any help I need, professionally or even personally. Among them, I would like to thank Amit and Abhinav for going through my work, my reports, reviewing, and helping me with my writing. I am grateful to Menzi for all the printout support whenever needed, including this thesis. The visit to Eindhoven would not be as much fun and fulfilling if not for the rest of the gang who I am forever thankful for. The idea of a PhD would not be as compelling to me without my fellow PhD friends, Shreya and Alok, who inspired me.

Also, it has been an experience to create a social life apart from the work life in Bergen. The void of the Eindhoven gang after I moved here has been filled by my salsa gang, in the Norwegian side. I would like to extend special thanks to Nhat Børnes for welcoming me into the salsa environment upon which my entire social circle in Bergen outside work is built upon. Apart, from being a friend he has been a guide to me in more than one ways. Their family including Nhat, Ingrid and My-Li have helped me integrate into the society here along with private Norwegian tuitions. I have been exposed to Norwegian culture and traditions through many friends in the circle, and particularly Iril. I am thankful to Iril for her effort on my Norwegian language and providing me an understanding of the local culture along with exposure to the food, also for the summary in Norwegian. I am also thankful to my gaming gang, Phoung, Erik, Shashi and Cathrine, who have instilled a new kind of gaming craze in me and make the weekends fun. I thank Linnea for all the good times, and Alejandro for taking me to all the good parties. Also, special mention of Daniel Schmidt for his immense interest in providing free salsa lessons in Fantoft which led me into this salsa circle.

Final part is dedicated to all my friends and family. In particular I am deeply grateful to my parents who have forever been supportive, guiding and understanding. I would not have come this far even, if not for them.

Summary

Wireless Sensor Networks (WSNs) consist of sensors and actuators operating together to provide monitoring and control services. These services are used in versatile applications ranging from environmental monitoring to industrial automation applications. Industrial Wireless Sensor Network (IWSN) is a sub domain of the WSN domain, focussing the industrial monitoring and automation applications. The IWSN domain differs from the generic WSN domains in terms of its requirements. General IWSN requirements include: energy efficiency and quality of service, and strict requirements are imposed on the quality of service expected by IWSN applications. Quality of service in particular relates to reliability, robustness, and predictability.

Medium Access Control (MAC) protocols in an IWSN solution are responsible for managing radio communications, the main consumer of power in every IWSN element. With proper measures, MAC protocols can provide energy efficient solutions along with required quality of service for process control applications. The first goal of the thesis was to assess the possibility of creating a MAC protocol exploiting properties of the application domain, the process control domain. This resulted in the creation of the Dual-Mode Adaptive Medium Access Control Protocol (DMAMAC) which constitutes the main contribution of this thesis. The DMAMAC protocol is energy efficient, while preserving real-time requirements, and is robust to packet failure. This has been guaranteed by the thorough evaluation of the protocol via simulation, verification, and implementation with deployment testing.

In parallel, we also investigated the possibility of using an alternative development approach for MAC protocols. Specifically, we have proposed a development approach based on MAC protocol model in CPN tools. The development approach consists of automatic code generation for the MiXiM simulation tool and the TinyOS platform.. We used the related GinMAC protocol as a running example for the development approach. The generated code for MiXiM simulation platform and the TinyOS implementation platform are evaluated via simulation and deployment respectively. This results in a faster design to implementation time, and closely related protocol artifacts, improving on the traditional approach.

Oppsummering

Wireless Sensor Network (WSN) består av sensorer og aktuatorer som opererer sammen for å gi overvåknings og kontrolltjenester. Disse tjenestene er brukt i allsidige applikasjoner med et spenn fra miljøovervåkning til industrielle overvåknings og automasjons applikasjoner. Industrial Wireless Sensor Network (IWSN) er et subdomene av WSN domene, med et fokus på overvåkning og automasjonsapplikasjonene. IWSN domene skiller seg fra de generiske WSN domene i form av sine krav. Generelt omfatter IWSN sine krav: energieffektivitet, kvaliteten på tjenesten som blir forventet av IWSN applikasjonene. Kvaliteten på tjenestene relateres spesielt til pålitelighet, robusthet og forutsigbarhet av de foreslåtte løsningene.

Medium Acess Control (MAC) protokoller i en IWSN løsning er ansvarlig for å administrere radiokommunikasjon som er den viktigste hovedforbrukeren av kraft i hvert av IWSN elementene. Med en riktig styring kan MAC protokoller gi energieffektive løsninger sammen med ønsket effekt av kvalitet for tjenesten av prosesskontrollapplikasjonen. Det første med denne oppgaven var å vurdere muligheten for å skape en MAC protokoll til å utnytte egenskapene til prosesskontroll domenet. Dette resulterte i en etablering av Dual Mode Adaptive Medium Acess Controll (DMAMAC), som er utgjør hovedbidraget til denne avhandlingen. DMAMAC protokollen er energieffektiv, samtidig som den bevarer sanntidskravet samtidig som den er robust mot pakkefeil. Dette har blitt garantert via en grundig evaluering av protokollen gjennom simulering, verifisering og gjennomføring testing.

Parallelt har vi også undersøkt muligheten fir bruken av en alternativ utviklet tilnærming for MAC protokollen. NtilnÃŠrmingrmere bestemt har vi foreslått en utviklet tilnærming basert på MAC - protokollens modell i CPN verktøyet. Denne utviklede tilnærmingen består av automatiske kodegenereringer for MiXiM simuleringsvertøyet og TinyOS plattformen. Vi brukte den relaterte GinMAC protokollen som en løpede eksempel for den utviklede tilnærmingen. Den genererte koden for MiXiM simuleringsplattformen og TinyOS gjennomføringsplattformen evalueres via en simulering og en henholdsvis distribusjon. Dette resulterte i en raskere design til implementeringstiden, en nærmere gjenstands protokoll og gav en bedring på den tradisjonelle tilnæmingen.

Contents

Part I

Overview

Chapter 1

Introduction

A Wireless Sensor Network (WSN) is a network comprised of sensor devices operating to monitor an environment of interest. The devices have three main responsibilities: sensing, processing of the obtained data; and communication with other devices. In some cases, the main processing of data happens at a sink node, which is a designated node in the network. A sink node is typically a computationally powerful wire-powered device in a WSN. A typical sensor device consists of a micro-controller, a sensor, and a radio transceiver. Application requirements for the WSN domain require these devices to be physically portable to assist in the calibration process, which is a part of the system setup. The portability requirement requires the devices to be small, and to be able to operate on batteries. The small size and battery-based operation results in limited energy and computing power. Initially, such devices were extensively used in military applications [1]. With the advancement in technology and extensive research in the domain of WSN, new opportunities and application domains have opened up [2, 3]. One of emerging domains of research is the Wireless Sensor Actuator Networks (WSAN) domain. A WSAN generally employs sensors and actuators to perform monitoring and control in an environment, across diverse operating elements. Actuators are devices responsible for controlling a physical quantity such as temperature, pressure, and flow. WSAN assists in building smart environments that employ automatic control systems for many application domains including smart homes, transportation, industrial automation, and healthcare.

In this thesis, we focus on industrial applications and Industrial Wireless Sensor Networks (IWSN) [4, 5, 6] which constitutes a sub-domain of

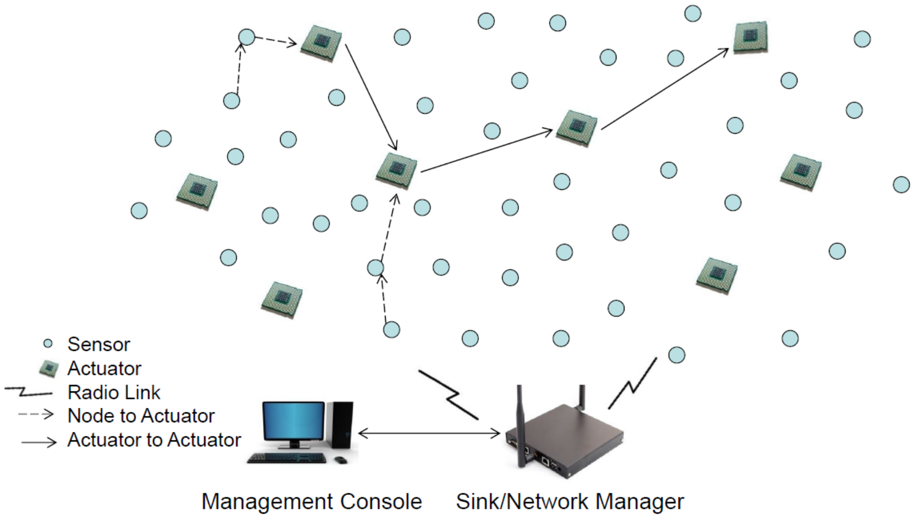


Figure 1.1.1: Wireless Sensor Actuators Networks

WSAN. IWSN applications have a set of requirements specific to the industrial setting [6, 7, 8]. Traditionally, industrial control automation applications have used wired communication. Major industrial players are now investing in research activities for IWSN, including the creation of standard solutions for process control automation applications, such as wirelessHART [9], ISA100.11a [10]. Lowering the cost while addressing the original requirements for quality of service is one of the main challenges when moving from wired to wireless networks.

1.1 Industrial Wireless Sensor Networks

An IWSN typically consists of sensors, actuators, a sink/network manager, and a management console to control the network as shown in Fig. 1.1.1. The sensor nodes are responsible for data collection and relay of packets towards the sink (network manager). The actuators obtain the processed data from the sink via multi-hop communication and routing. Alternatively, the sensors may also communicate (information) directly to the actuators as illustrated by the dashed arrows in Fig. 1.1.1. The network manager is also responsible for scheduling and managing the network, and the management console is used to control the network manager. A typical sensor/actuator node for the industrial domain implements four important functions of IWSN

[7]: *Routing, Medium Access Control (MAC), Transport,* and *Security.* Multiple standards have been developed to provide solutions for IWSN applications. The most important ones are: Zigbee [11], ISA100.11a [10], WIA-PA [12], and wirelessHART [9]. Below we provide a broad classification of industrial applications, requirements specific to use in the industrial domain, and current standardization efforts to provide efficient IWSN solutions.

1.1.1 Applications of IWSN

WSNs are applied in a wide range of industrial applications and is not only replacing existing wired solutions, but also creating new and effective solutions. Industrial applications include operation in harsh environments such as having high temperature apparatus close to the sensor devices and actuators. Eliminating the requirement of manual intervention and reducing the size and cost of the devices is a main focus points of emerging wireless solutions.

Monitoring systems Initially, WSNs were used in military applications for monitoring purposes, and later expanded to additional applications areas. These new applications involve two types [8]: environmental monitoring and condition monitoring. Environment monitoring includes monitoring of physical quantities in a given environment and collecting this information over a period of time. Condition monitoring includes monitoring of equipments in the industrial setup for noise and vibration. An application example for condition monitoring on a smart grid can be found in [13] which is concerned with monitoring of components critical for operation of a smart grid. The monitoring process looks for failures or erroneous behaviours. An example of remote monitoring in the oil and gas industry can be found in [14] for pipeline monitoring.

Safety systems While working in harsh industrial environments, safety is an important concern. Fire detection [15] systems, methane gas leak detection systems [16] are examples of WSN applications. These systems continuously monitor a given environment for events that threaten safety and when an event occurs, authorities are notified for further handling. Another gas leak detection example involving hazardous gases can be found in [17]. Depending on the seriousness of the threat, the requirements for the safety system applications vary. Real-time requirements, including low-delay, high-reliability, and robustness constitute a general set of requirements for these systems.

Control systems An important part of process automation applications are control systems. Control systems can be broadly classified [10] into: *open loop* and *closed loop* systems. The *open loop* system consists of a human in the loop that manages and controls the process. The *closed loop* is a completely automated system, which could have a human as a supervisory controller. Process control systems generally have strict real-time requirements typically in the order of milliseconds for delay. They also have high reliability requirements which is essential to keep the process being controlled stable and operational. The set of requirements imposed by the control systems in general pose a challenge to the wireless solutions. A wireless temperature control network is presented in [18], and [19] discusses a water pump control system.

In this thesis, we focus specifically on applications of IWSN in control systems [20].

1.1.2 IWSN Requirements

The IWSN domain is a subset of the WSAN domain and has requirements specific to the industrial domain [6, 7, 8]. Below we discuss the requirements, specifically focussing on the control system applications.

Energy efficiency IWSN devices are generally battery powered, and due to their limited size, the battery capacity is also limited. Energy consumption can be viewed in two dimensions as mentioned in [7]: *low energy consumption* and *efficient energy consumption*. *Low energy consumption* solutions focus on keeping the energy consumption at every single node low. *Efficient energy consumption* solutions focus on balancing the energy consumption throughout the network to extend the network lifetime.

Quality of Service (QoS) Every application defines a set of performance metrics based on its requirements. These metrics are used to assess if the proposed solutions meet the application requirements. For example, real-time process control in industrial applications has strict requirements on delay (low), reliability (low packet loss), dependability, robustness, and predictability. A protocol solution attempts to obtain an optimal trade-off between energy efficiency and providing the required QoS.

Wireless channel conditions are dynamic and vary with time. This is particularly true given the influx of multiple types of wireless services used in an environment in general and also within the particular application domain. For example, WiFi, mobile services and other electromagnetic noise creates interference with the operation of the IWSN. Thus, protocols need to employ techniques to be robust under potentially deteriorating channel conditions. Predictability allows the application users to create definitive models for the operation of IWSN solution and is one of the important IWSN application requirements [21, 22].

1.1.3 Industrial Standards

The IEEE 802.15.4 [23] is an IEEE standard to support low-power, low-data rate radio applications with a short coverage area. It was developed as a standard by the Personal Area Network (PAN) working group. This standard has been widely used for WSN solutions proposed by industrial working groups such as WirelessHART [9] and ISA100.11a [10]. It mainly specifies the MAC and the physical layer. IEEE 802.15.4 uses Carrier Sense Multiple Access (CSMA) with Collision Avoidance (CA) and Guaranteed Time Slots (GTS). The 802.15.4 PHY (Physical layer) is embedded into transceivers such as the CC2420 [24]. Later, given the industrial requirements of predictability, reliability, and real-time aspects, the IETF working group on Wireless Personal Area Networks (WPAN) proposed a new version called IEEE 802.15.4e [25, 26]. The new standard aims at addressing co-existence issues using Time Slotted Frequency Hopping (TSFH). Additionally, other features for e.g., dedicated and shared slots, multi-channels, were proposed [26, 27] to improve the MAC performance.

WirelessHART [9] is the wireless standard developed by the HART Communication Foundation (HCF) for process automation applications. The network architecture proposed by HCF consists of multiple components including: *field device, adapter, handheld device, gateway, network manager,* and *security manager*. The *field device* is a field instrument, either a sensor or an actuator device. A *handheld* device is a portable device to collect data, configure, and diagnose field devices. The *gateway* connects the plant automation network to the network manager, and the security manager. The architecture relies on centralized control with a dedicated device called the *network manager* for control and management. *The security manager* is responsible for the security in the network. The physical layer in wirelessHART is based on 802.15.4 PHY, but the MAC in WirelessHART is significantly different and is based on the Time Synchronization Mesh Protocol (TSMP). TSMP is a Time Division Multiple Access (TDMA) protocol developed by Dust Networks [28]. It uses TDMA and applicable features from IEEE 802.15.4 MAC. TDMA is reliable in comparison to random access techniques, and reliability is one of the important requirements in process automation applications. To overcome radio frequency interference [29, 30], the standard employs channel blacklisting, where poor frequency channels among the 16 available channels are blacklisted based on observed performance.

The International Society of Automation (ISA) [31] set out to develop its own standard for wireless networks in process automation, similar to the standardization process by HCF. The standard created is ISA100.11a [10]. ISA is based on the IEEE 802.15.4 standard similar to wirelessHART and shares some common features in the MAC layer, like employing TDMA techniques. In addition, it also uses the CSMA-CA techniques similar to the base IEEE 802.15.4 MAC. CSMA-CA is used to obtain higher utilization. ISA100.11a also uses channel hopping techniques to address interference issues. ISA100.11a supports the IPv6 standard, enabling support for the new domain referred to as Internet of Things (IoT) [32]. The ISA100.11a standard uses a wired backbone network along with a gateway, system manager (network) and security manager similar to wirelessHART to manage the network. It also has support for integrating legacy protocols into the ISA solution, e.g., wired HART and interoperability with networks based on wirelessHART.

It is important to note that all these standards constitute more of a framework than a complete solution. The frameworks provide flexibility for developers to include their own software by modifying an existing framework to

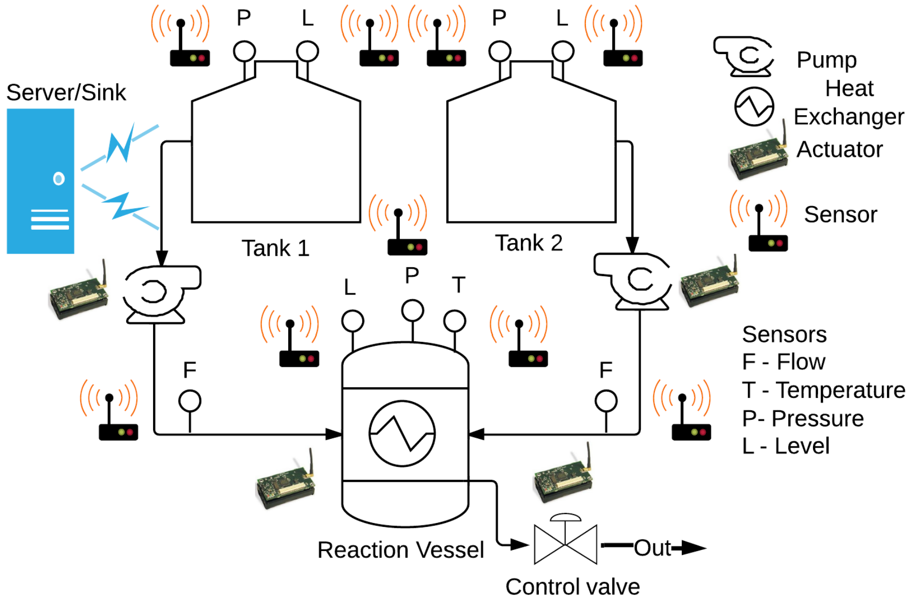


Figure 1.2.1: An example process control scenario

match specific requirements. As an example, existing MAC protocols in the framework can be extended to include additional features addressing application specific requirements. For a more detailed comparison between different wireless standards, including wirelessHART and ISA100.11a, we refer to [33].

1.2 Process Control Application

In this thesis, we focus on the use of IWSN for the process control applications. An example of a representative process control system is shown in Fig. 1.2.1. This is similar to the example process control system used in the case study for the GinMAC protocol [34]. It shows a chemical process plant with sensors and actuators installed to control the chemical process. The example includes two storage tanks (Tank 1 and 2) to store chemicals to be mixed, in the reaction vessel. The actuators control the flow of chemicals from the storage tanks to the reaction vessel based on the actuation commands from the sink/network manager. The sink makes control decisions based on the sensor readings it obtains and the requirements of the process being controlled. Multiple types of sensor units are used in the scenario including: flow, temperature, pressure, and level sensors. Three types of actuators are used: pump

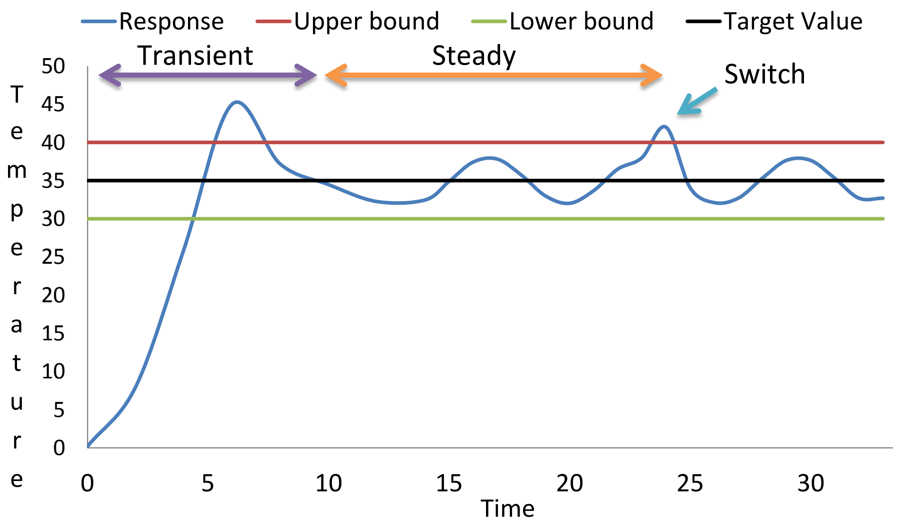


Figure 1.2.2: Chart representing of process control variables

controller (for flow/level), heat exchanger (for temperature), and a control valve (for pressure/level). With the assistance of these sensors, actuators, and the underlying *process model*, the sink performs monitoring and control. Process models are models representing the process in operation and the control procedure. In other words, the process models provide the decision logic used by the sink to process the incoming sensor data. The processed data is then relayed to the actuators as actuation commands for required operation.

A typical process control operation could result in a graph similar to the one shown in Fig. 1.2.2. It shows a measured physical quantity, e.g., temperature monitored and controlled over time. Process control normally consists of the two states *steady* and *transient* and the process control alternates between these two operational states. The *transient* state is a state in which there is rapid change in a measured physical quantity (temperature) as indicated in the first section of the graph. An IWSN solution has to communicate large amounts of data in the transient state in order to control the process without disruptions. The *steady* state in contrast to the transient state has a small rate of change. The measured value (in this case temperature) generally lies within a bounded interval for the steady state as shown in the second part of the graph. This allows for the IWSN to reduce the data traffic towards the sink from the sensors.

1.3 Functions

An IWSN solution for process control applications is made up of a suite of protocols that provide the different functions required. The suite of protocols implement four important functions:*Transport, Routing, Medium Access Control (MAC)* and *Security*. These in addition to an application layer make up a complete solution for a given industrial application. Below we discuss Transport, Routing, MAC and security functions.

Transport. In the context of this thesis, reliability is an important requirement for the design of industrial solutions. Reliability here refers to the reliable delivery of data from source to destination, thus reducing packet loss at the same time as addressing real-time delay requirements. Packet loss can be caused either by congestion or interference that exists in wireless channels. Transport protocols are generally responsible for reliable communication in an IWSN in conjunction with the MAC protocols. Transport protocols achieve this goal by congestion control/avoidance and implementing features to reduce interference effects. The application requirements can further specify the reliability requirements at a finer granularity based on the goals it needs to achieve as described in [7]. Example transport protocols applicable in the IWSN domain include the Asymmetric and Reliable Transport Mechanism (ART) [35], the real-Time and reliable transport protocol (RT^2) [36], and the Reliable Bursty Convergecast (RBC) [37] protocol.

Routing. In a typical multi-hop network, multiple paths exists between a given source and destination. The routing layer in an IWSN solution is responsible for efficiently routing data within the network by finding the best path suitable for the application requirements. Important design requirements for routing protocols include: *fault tolerance, energy efficiency, load balancing, service differentiation,* and *Expected Transmission count(ETX)* [7]. Apart from the general design requirements, the planned topology of the network imposes additional requirements to the design. Example protocols applicable in the domain are: the Multipath Multi-SPEED protocol (MM-SPEED) [38], the Threshold Sensitive Energy Efficient Sensor Network protocol (TEEN) [39], and the Routing Protocol over Low-power and Lossy

Networks (RPL) [40].

Medium Access Control (MAC). MAC protocols are mainly responsible for controlling the medium access in the network. Thus, it is directly responsible for controlling the radio, which is the largest consumer of energy in a sensor-actuator device [41, 42]. The MAC protocol decides the required schedule for the network operation based on the application requirements. Two important types of approaches used in MAC protocols are Time Division Multiple Access (TDMA) and Carrier Sense Multiple Access (CSMA). Protocols implementing the TDMA approach divide the available time between the nodes in the network. Nodes operating as part of a CSMA-based protocol first sense the channel whenever they have data to be sent. If the channel is found to be clear with no communication in the vicinity, the nodes go ahead with data communication. Some protocols use both TDMA and CSMA concepts to realise a hybrid operation [43]. This allows the protocol designers to exploit the advantages of the two approaches. Some example MAC protocols that could be applicable in the IWSN domain are the GinMAC protocol [44], QoS aware MAC [45], Power Efficient and Delay Aware MAC for Sensor Networks (PEDAMACS) [46], and Emergency Response MAC (ER-MAC) [47].

Security. Wireless networks are inherently susceptible to security attacks due to the lack of physical connections. Additionally, given the substantial difference in the requirements of the IWSN applications from traditional wired networks, the security techniques applied to wired networks are not directly applicable [48]. Given the limited capability of the devices included in the network, developing a lightweight but strong security procedure is a challenge in itself. The wireless standards wirelessHART and ISA100.11a use dedicated devices as security managers.

1.4 Research Questions

Given our focus on specific industrial requirements, and the fact that MAC protocols control most of the energy expenditure, including its ability to affect other requirements (e.g. delay), we further narrow our focus on MAC protocols for the IWSN domain. The research questions that this thesis addresses are listed below:

- Can MAC protocols for wireless process control be made energy efficient by exploiting intrinsic properties of the process control domain?
- What guarantees can be provided with respect to real-time operation and reliability?
- How can model-driven engineering techniques be used for MAC protocol development?

1.5 Goals and Contributions

The main contribution of the thesis is the proposed Dual-Mode Adaptive Medium Access Control Protocol (DMAMAC). The DMAMAC protocol is proposed for process control application with real-time requirements. The proposed protocol is then thoroughly analyzed and refined to arrive at robust, energy efficient protocol with two variants (Hybrid and TDMA). The analysis process included validation, verification, simulation and deployment test. Prior to the protocol proposal, to get an idea about the state-of-the-art in the domain, we investigated state of the art as a first step towards answering the research questions. This investigation resulted in identifying the MAC protocols applicable to the domain of research, the process control domain in particular. Further, we studied the identified protocols thoroughly to be able to extend them in order to exploit the domain properties to result in design that performs better than the existing solutions. We identified the GinMAC protocol [44] as a base and proposed the DMAMAC protocol. In parallel, we also studied state-of-the-art model-driven techniques for the design of communication protocols. We used one of the existing tools, the PetriCode tool [49], and proposed a design and development approach on model-driven principles.

We proposed an initial version of the DMAMAC protocol in the article "Towards a dual mode adaptive MAC protocol (DMA-MAC) for feedback-based networked control systems" [50]. The paper mainly discussed the

DMAMAC-Hybrid variant. Additionally, we evaluated the energy consumption of the DMAMAC protocol against the related GinMAC [44] protocol analytically, and showed that it provides substantial reduction in energy consumption for target configurations.

The DMAMAC protocol was then evaluated via simulation and was presented in the article [51]. Based on the initial simulation evaluation, a fully-TDMA version was derived from the DMAMAC protocol to obtain a DMAMAC variant with improved reliability and predictability. Both variants of the DMAMAC-protocol were compared for several metrics including energy efficiency, network lifetime, and switch failure probability (notification packet failure).

The contribution of the article "Model-based Specification and Validation of the DMAMAC Protocol" [52] is to describe the validation and verification of the DMAMAC protocol via model checking. We describe the modeling for both DMAMAC protocol variants using the Uppaal tool [53]. Models created are validated and verified using queries. The queries include: configuration-specific, configuration-independent and real-time queries.

The DMAMAC protocol is implemented and evaluated via deployment in the article "Implementation and Deployment Evaluation of the DMAMAC Protocol for Wireless Sensor Actuator Networks" [54]. We, mainly focus on the goals drawn out for the implementation evaluation of the DMAMAC protocol. The DMAMAC protocol nesC implementation code is deployed using a hardware platform (Zolertia), and evaluated in a realistic industrial application scenario.

The contribution of the article "Towards a model-based development approach for wireless sensor-actuator network protocols" [55] is to present an initial idea of a Model-Driven Software Engineering (MDSE) [56] approach for MAC protocols. We present the MDSE approach based software code generation architecture. The entire process uses: a modeling tool, PetriCode for code generation based on platform specific code templates (simulation and deployment). An extended version is presented in "Model-based Development for MAC Protocols in Industrial Wireless Sensor Networks" [57]. The simulation platform source code generated is simulated in the MiXiM platform and analyzed for performance. The implemented code for deployment is deployed using a hardware platform and evaluated for link quality.

We use the GinMAC protocol as a running example to present the MDSE approach.

1.6 Outline

This thesis is divided into two main parts: Part I provides an introduction and overview of the dissertation, and Part II is comprised of a collection of articles [50, 51, 52, 54, 55, 57] that make up the dissertation. Part I in addition to the current chapter, is divided into the following:

Chapter 2 provides an introduction to the DMAMAC protocol and the articles concentrating on the design of DMAMAC. We discuss the work related to the DMAMAC protocol, followed by a detailed overview of the features of the DMAMAC protocol. We also discuss the two variants of the DMAMAC protocol, DMAMAC-Hybrid and DMAMAC-TDMA along with their differences and applications.

Chapter 3 discusses the evaluation of the DMAMAC protocol. The performance evaluation of the DMAMAC protocol is done via a network simulator platform focussing on energy efficiency and network lifetime. The verification of the design of the DMAMAC protocol is discussed as a second part of the evaluation.

Chapter 4 discusses the prototype implementation and deployment of the DMAMAC protocol. In addition to providing an overview of the implementation article, we add the details related to the implementation setup missing in the article itself.

Chapter 5 In the chapters 2, 3, and 4, a traditional design and development methodology is followed. In chapter 5, we provide an overview of the alternative development methodology that we have suggested. The alternative development methodology provides a model-based approach for design and development of MAC protocols. As a running example, the GinMAC protocol is used.

Chapter 6 We discuss conclusions and possible future directions in the last chapter. Additionally, a summary of the overview part, and applicability of the DMAMAC protocol beyond the IWSN domain is discussed.

In addition to summarising the contribution of each article, we also put the limitation of the work into perspective.

Part II is comprised of the following published/submitted articles:

[50] A. Ajith Kumar S., Knut Øvsthus, and Lars Michael Kristensen, "Towards a Dual Mode Adaptive MAC protocol (DMA-MAC) for Feedback-based Networked Control Systems", in Proceedings of the 2nd International Workshop on Communications and Sensor Networks, Procedia Computer Science, 34, 505 –510, 2014.

[51] A. Ajith Kumar S., Lars M. Kristensen, and Knut Øvsthus, "Simulation-based evaluation of DMAMAC: A Dual-Mode Adaptive MAC protocol for process control", in Proceedings of the 8th International Conference on Simulation Tools and Techniques, pages 218 –227, 2015.

[52] A. Ajith Kumar S., Andreas Prinz, and Lars M. Kristensen, "Model-based verification of the DMAMAC protocol for real-time process control", in Proceedings of the 9th International Workshop on Verification and Evaluation of Computer and Communication Systems, 1431:81 –96, 2015.

[54] A. Ajith Kumar S., Knut Øvsthus, and Lars M. Kristensen, "Implementation and Deployment Evaluation of the DMAMAC Protocol for Wireless Sensor Actuator Networks", in Proceedings of the 7th International Conference on Ambient Systems, Networks and Technologies, Procedia Computer Science, 83, 329 –336, 2016.

[55] A. Ajith Kumar S. and K. I. F. Simonsen, Towards a model-based development approach for wireless sensor-actuator network protocols, in Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems, pp. 35 –39, ACM, 2014.

[57] A. Ajith Kumar S. and K. I. F. Simonsen, Model-based Development for MAC Protocols in Industrial Wireless Sensor Networks, in Proceedings of the International Workshop on Petri Nets and Software Engineering (Accepted), 2016.

The reader of this thesis is assumed to have knowledge about communication systems and the use of wireless technologies in general. The artifacts for simulation code and verification models (+code) can be found in [58].

Chapter 2

The DMAMAC Protocol

This chapter is based on the article article "Towards a dual mode adaptive MAC protocol (DMA-MAC) for feedback-based networked control systems" [50]. The article mainly provides an initial specification of the Dual-Mode Adaptive MAC (DMAMAC) protocol, along with underlying assumptions. The article gives an introduction to the process control application for which the DMAMAC protocol was proposed. Further, it proposes the dual-mode mechanism in the DMAMAC protocol, a detailed description of each slot and its function in the two types of superframes (steady and transient). Also, an introduction to the idea of the alert message mechanism to facilitate the mode-switch. The article mainly discussed the DMAMAC-Hybrid variant. Additionally, we evaluated the energy consumption of the DMAMAC protocol against the related GinMAC [44] protocol analytically, and showed that it provides a substantial reduction in energy consumption for target configurations.

The DMAMAC protocol was first proposed in [50] targeted at process control applications. After simulation analysis of the initial version, an improved variant was proposed and analyzed in [51]. The two variants use Time Division Multiple Access(TDMA) and Carrier Sense Multiple Access (CSMA) methods for channel access. The two variants are called DMAMAC-Hybrid (TDMA+CSMA) and DMAMAC-TDMA Protocol. In addition to providing an overview of the DMAMAC protocol along with its variants, we also discuss the differences between the two, their pros and cons. It should be mentioned that the DMAMAC protocol can be used within the framework of the industrial standard solutions presented in 1.1.3.

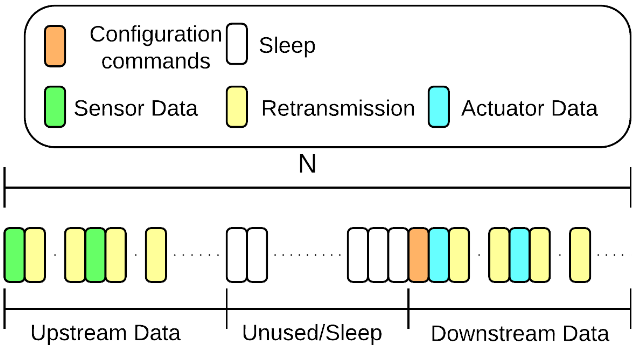


Figure 2.1.1: A generic GinMAC superframe

2.1 MAC Protocols for Process Control

The GinMAC protocol [44] was proposed for the GINGSENG [59] project aimed at performance monitoring and control in Industrial Wireless Sensor Networks (IWSN). The design of the GinMAC protocol aims at satisfying real-time requirements of industrial applications. GinMAC is a TDMA-based protocol with features that include: *offline dimensioning*, *exclusive TDMA*, and *delay conformed reliability control*. *Offline dimensioning* means that the scheduling of the superframe for the network using the GinMAC protocol is calculated offline prior-deployment. *Exclusive TDMA* means that the slots used by the nodes are exclusive and cannot be re-used by other nodes. GinMAC also assumes data transmission of maximum packet length and associated acknowledgement within the same slot. The GinMAC authors propose pre-deployment assessment of the channel conditions in the deployment environment. Based on results obtained from the conducted experiments, and delay requirements of the application, *delay conformed reliability control* is employed to decide the number of re-transmissions slots.

A typical GinMAC superframe is shown in Fig. 2.1.1. Apart from the essential sensor and actuator data slots, the GinMAC superframe incorporates re-transmission for both types of data slots, sleep slots, and a configuration command slot. The configuration command slot is used for transmitting management information by the sink, such as time synchronization messages. The network topology for the GinMAC protocol is a tree topology as shown in Fig. 2.1.2. GinMAC protocol is an important background and related work, which will be used extensively in the rest of the thesis.

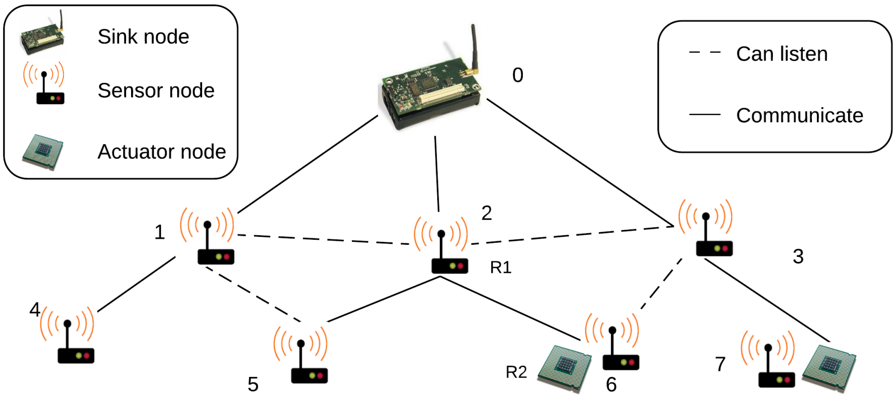


Figure 2.1.2: The network topology for GinMAC

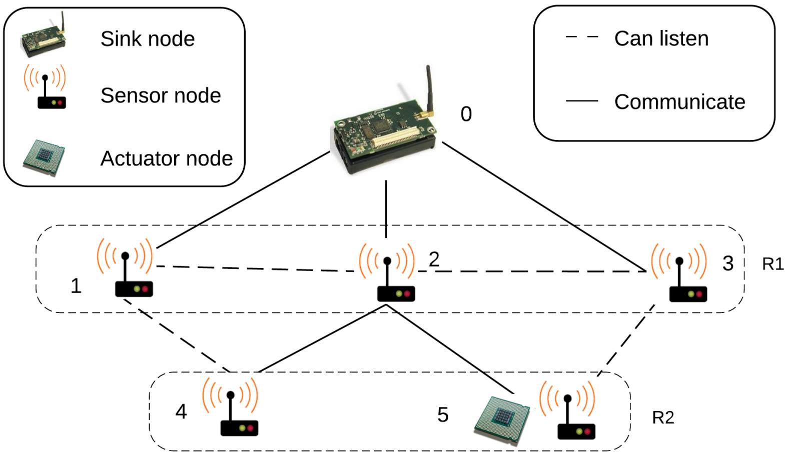


Figure 2.2.1: An example node topology for the DMAMAC protocol

2.2 The DMAMAC Protocol

The DMAMAC protocol is designed for a tree topology network similar to GinMAC. The network consists of sensors, actuators, and a sink. Both the sensors and the actuators are equipped with wireless communication capable transceivers. The sensors measure a given physical quantity (e.g., temperature) over a certain geographical area. The actuators act on incoming commands influencing a certain physical quantity, e.g., influencing temperature by heating up or cooling down. The sink is a computationally powerful wire powered device, it performs data processing on the sensed data and communicates processed data to the actuators. The sensors and actuators are battery powered, and have relatively lower computational power.

The network is organized into levels, where each level is associated with a rank *Rn* where $n = 1,2,3..$ as shown in the Fig. 2.2.1. The only difference between the topology presented previously in Fig. 2.1.2 is the indication of levels associated with a rank required by the DMAMAC protocol. R1 and R2 in the case of Fig. 2.2.1. The rank is used in the alert message mechanism of the DMAMAC-Hybrid variant of the protocol. The DMAMAC protocol serves the dual-states of the process control via two separate operating modes in the protocol. The transient state of the process control is served by the transient mode in the DMAMAC protocol, and the steady state is served by the steady mode of the protocol.

Transient Superframe The transient superframe is designed for the transient mode of the DMAMAC protocol, and is based on the GinMAC [44] protocol superframe. The design of the transient superframe was first proposed in [50] and has remained unchanged. This is unlike the structure of the steady superframe. The main goal of the transient superframe is to support a high data rate in order to support the transient state of the process control. Additionally, it provides reliability measures by providing additional slots. When the packets fail due to poor channel conditions, they are sent again in the additional slots. The structure of the transient superframe is shown in Fig. 2.2.2. The number of re-transmission slots is a configurable parameter in the DMAMAC protocol design, and can be set based on application requirements. For an industrial environment with high packet loss, the application could benefit from providing more than one re-transmission slot. Also, this can be configured separately for sensor data and actuator data. The transient superframe is the same for both variants of DMAMAC (hybrid and TDMA). The length of a transient superframe is denoted as Nt.

Steady Superframe The steady superframe is designed for the steady mode of the DMAMAC protocol. The steady mode is designed to support the steady state of a process control application which has low-data rate requirements. This allows for incorporating a larger number of sleep slots than in the transient superframe. The steady superframe is designed such that its length (denoted as Ns) is a multiple of Nt, i.e, $Ns = n \times Nt$ where $n \geq 1$ is a configurable parameter that can be set based on application requirements.

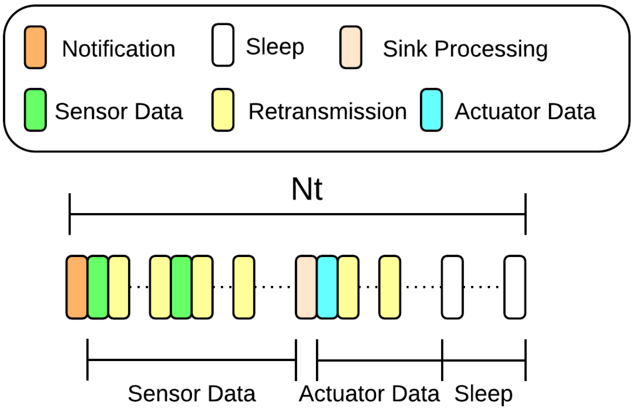


Figure 2.2.2: Transient mode superframe [50]

Essentially, the higher the value of n, the larger is the energy savings in the steady state. The energy saving is also dependent on the occurrence of the steady superframe during the operation. Also, Ns being a multiple of Nt allows seamless synchronization for nodes between the operating modes. For example, if a node misses a notification packet with mode switch command (steady to transient), it still will receive all the forthcoming notification packets and can switch to the transient mode at the beginning of the next transient part (in the steady superframe).

The steady mode superframe was first proposed in [50] and further modified in [51] based on performance analysis results. The first modification included moving the notification slot from the beginning of the second Nt part of the steady superframe to the end, and also including it for each Nt parts that follows. Moving the notification to the end, after the alert slots, ensured quick state change after an alert is received. In [54], we restored the notification slot to the beginning of each Nt slot to improve robustness against packet failures. This is explained in detail in [54]. Thus, the final version of the steady superframe shown in Fig. 2.2.3 includes two notification slots in each Nt part as opposed to one in the initial version. Additionally, the steady superframe has two configurable parameters. One is the number of re-transmission slots per transmission slots; the other is the number of alerts slots.

Alert messages An important part of the dual-mode DMAMAC protocol is the mode switch procedure. The switch between the transient and steady

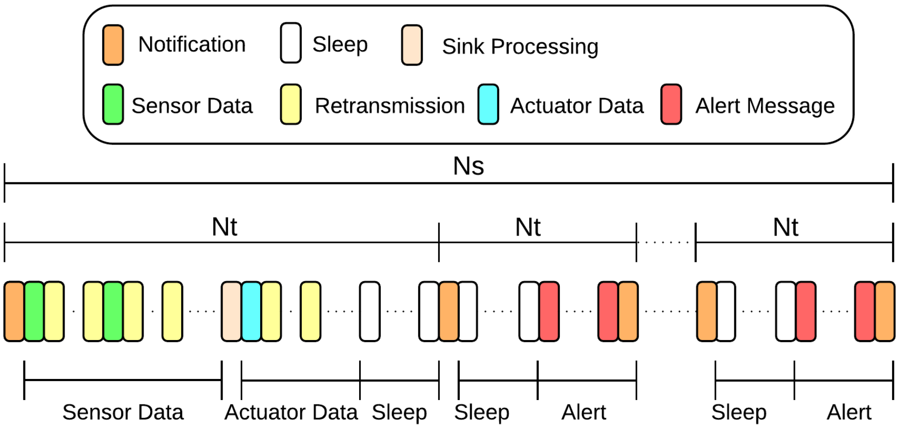


Figure 2.2.3: Steady mode superframe

modes are facilitated by different actors (sink or sensors) depending on the nature of the switch. The switch from transient to steady is facilitated by the sink based on an assessment of the process control state. This could be automatic statistical analysis or manual setting based on deployment experience. This switch is less critical compared to the switch from steady to transient. The switch from steady to transient is initiated by the nodes and implemented network-wide by the sink. This is a critical switch since the transient state requires transient mode support from the protocol in order to accommodate a high data-rate. This switch is decided based on a threshold interval set by the sink. When the measured physical quantity breaches the aforementioned threshold, the sensor nodes notify the sink of this breach via alert messages. The alert messages are sent in the alert slot (see Fig. 2.2.3). The scheduling of the alert slots is the main difference between the two variants of the DMAMAC protocol (hybrid and TDMA) and is discussed in the following section.

2.3 DMAMAC-Hybrid and DMAMAC-TDMA

The first proposed variant of the DMAMAC protocol was the DMAMAC-Hybrid protocol. The term hybrid comes from the fact that it combines CSMA and TDMA within the steady superframe. The transient superframe uses entirely TDMA and the steady superframe uses CSMA for the alert message slots. The use of CSMA for alert message slots results in a scalable

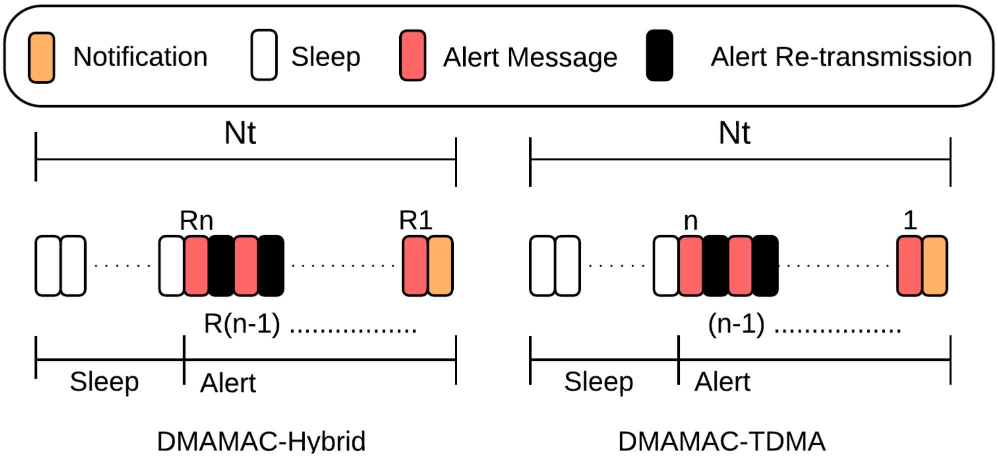


Figure 2.3.1: Alert message mechanism in the DMAMAC protocol, DMAMAC-Hybrid (a) and DMAMAC-TDMA (b)

steady superframe and energy conservation. The slot distribution is based on the rank in the tree topology as discussed in [52]. Each rank has two specific slots assigned: one for reception (RX) and one for transmission (TX). The alert mechanism for DMAMAC-Hybrid is shown in Fig. 2.3.1(a). The slot identification with *Rn* indicates slot per rank for the DMAMAC-Hybrid variant and re-transmission slots (if any) follow the main transmission slot. The actual alert sending mechanism was modified in [51] from the initial version proposed in [50] based on the simulation performance analysis done in [51]. Initially, we chose to send two alert messages in the same alert slot. Duplication was used to prevent loss of alert messages due to collision.

Further, we tried alternative configurations which proved to perform better based on simulation analysis, resulting in the change to initially proposed model. In the final version of the protocol, a node performs the following steps when an alert has to be sent and the node is in the alert transmission slot. Firstly, a random delay is chosen by the respective sensor node, and it waits for this duration in its alert slot. At the end of the delay, the node performs carrier sense to check if there is any other alert transmission taking place in the vicinity. In the absence of other alert transmissions, the sensor node transmits its alert message to its parent in the tree topology.

The DMAMAC-TDMA has a simpler alert message approach than the hybrid version. The alert mechanism for DMAMAC-TDMA is shown in Fig. 2.3.1(b). As the name suggests, each sensor node has its own alert message transmission slot. It forwards its alert message to its parent, which in turn forwards it towards the sink. Note that only one alert message is required to make the switch, and thus the parent nodes only send one alert message, which could be either an alert to be forwarded or their own alert message. No separate alert message forward slots are used. The slot identification with node identifier $(n, n-1....)$ is for the DMAMAC-TDMA variant with subsequent re-transmission slots as shown in the Fig. 2.3.1(b).

As a configurable parameter, the re-transmission slots could be used for alert messages too, to make the protocol more robust in poor channel conditions. This is true for both DMAMAC-Hybrid (per-level re-transmission slots) and DMAMAC-TDMA (per node re-transmission slots). The alert message slots are shown in Fig. 2.3.1 for both DMAMAC-Hybrid (left) and DMAMAC-Hybrid (right) with optional re-transmission slots. The number of re-transmission slots (black) may vary depending on the requirements.

The two variants of the DMAMAC protocol have their own advantages and disadvantages in the following aspects:

Energy efficiency In certain cases the DMAMAC-TDMA protocol could consume more energy than the hybrid variant. This is specifically true due to alert messaging. As an example, consider a case where a large number of nodes exist in the network and each parent node has three child nodes. The parent node has to have a separate receive slot for each child node. In the case of hybrid, the parent would only have one alert receive slot, but the same node in the TDMA variant would have three alert receive slots. This means that the energy consumption for its alert slot is three times that of the hybrid variant alert slot.

Packet failure DMAMAC-Hybrid may suffer from packet collisions in its CSMA part for alert messaging. This affects the performance of the critical switch from steady to transient. DMAMAC-TDMA has an advantage over the hybrid variant in this case, but could still be affected by packet failure due to poor channel conditions (which affects both variants equally).

Switch delay The relatively low number of alert slots in the hybrid variant facilitates low-switch delay compared to the TDMA variant. This could be used as an advantage while designing smaller superframes facilitating low-delay.

Predictability The TDMA variant has a higher level of predictability due to the use of entirely TDMA. The hybrid variant suffers from the non-determinism introduced in its alert message slots and due to collision.

As discussed above, both variants of the DMAMAC protocol can be useful depending on application requirements. Thus, it is up to the application designer to use the appropriate variant based on the application requirements. Additionally, the configurable parameters provide flexibility to adapt the performance of the protocol to the application requirements. In the end, the choice of the protocol to be used depends on the type of trade off that the application can have, essentially, the trade-off between energy efficiency and predictability.

2.4 Related Work and Perspectives

The Breath [60] protocol suite was proposed as a complete protocol stack with routing, MAC, and radio controlling mechanisms included. Breath is also aimed at industrial control applications using WSN similar to the DMAMAC protocol. The main focus of the Breath protocol stack at the MAC layer is to have a CSMA/CA based solution with clustering techniques. The key assumption underlying the protocol is that the application has a high amount of randomly occurring events. The protocol aims to adapt to these random events while being energy efficient at the same time. In contrast, we on the other hand focus on applications that require predictability, energy efficiency along with satisfaction of real-time requirements.

The Priority enhanced Medium Access Control (PriorityMAC) [61] is a protocol designed for IWSN. The focus of the protocol is to facilitate critical traffic by separating data into different traffic classes. The main classes are: critical, secondary and low priority. It successfully achieves low-latency for the critical priority traffic (critical) by higher bandwidth allocation. The authors have analyzed the performance of the protocol via simulation in MATLAB and further implemented the protocol on TinyOS and deployed it on a testbed for further analysis. With respect to the DMAMAC protocol, the focus of PriorityMAC is entirely different. The focus of the PriorityMAC protocol is to classify data into different traffic classes, with critical data being assigned higher priority to provide reliable service. It could be possible that these two protocols could co-exist, i.e., the DMAMAC protocol data could benefit from priority classifications and usage of PriorityMAC features.

Z-MAC [43] is a hybrid protocol using both CSMA and TDMA. Z-MAC is a distributed protocol (for slot selection) and also a dual mode protocol similar to the DMAMAC protocol with two modes of operation: low-contention level and high contention level. Contention here refers to contention for channel resources, i.e., number of nodes competing for using the channel at a given time. The nodes running on Z-MAC list their two-hop neighbors and locally perform scheduling in a distributed manner such that no two nodes select the same slot. CSMA is used for low-contention level duration and TDMA for high contention level to obtain high bandwidth utilization. The scheduling in Z-MAC is dynamic, whereas in the DMAMAC protocol, we use offline scheduling, where the node slots are calculated pre-deployment similar to the GinMAC protocol. The offline scheduling in combination with TDMA increases the predictability of the DMAMAC protocol operation, which is an important requirement of IWSN.

The GinMAC [44] protocol, used as a basis for the DMAMAC protocol is also used for monitoring and control. The DMAMAC protocol is energy efficient compared to the GinMAC protocol for process control scenarios with dominant steady state as presented in the analytical evaluation in [51]. Also, the longer the length of the steady superframe the more is the energy savings.

This chapter mainly provided an overview to the DMAMAC protocol design based on details and excerpts from [50] and [51]. Apart from these, the steady superframe modification made in [54] is also mentioned, which led to the final version of the steady superframe. Below, we place the article [50] in perspective based on the current standing of the DMAMAC protocol.

The DMAMAC protocol is aimed at a section of process control applications, where the steady state dominates the time of execution duration. Thus, the protocol is assumed to operate in steady mode for most of the execution duration, consuming lesser energy relative to the GinMAC protocol. For configurations with higher transient state occurrence in the process control operation, users can benefit from using the GinMAC protocol. The DMAMAC protocol has an increased operation and implementation complexity compared to the GinMAC protocol. We would like to also stress on the fact that, the DMAMAC protocol has been successfully improved since its first proposal in [50].

Chapter 3

Protocol Evaluation

The DMAMAC protocol was thoroughly simulated and also verified to obtain higher confidence on the specification of the DMAMAC protocol. In this chapter, we discuss the simulation and verification of the DMAMAC protocol, based on articles [51] and [52].

The contribution of the article [51] is mainly the simulation evaluation of the DMAMAC protocol. Multiple simulation runs for different configurations provided better insight into the performance of the protocol. In particular, the alert message mechanism of the protocol (DMAMAC-Hybrid) was improved for better performance. Also, a fully-TDMA version was carved off the DMAMAC protocol to obtain a DMAMAC variant with improved reliability and predictability of the protocol. Both variants of the DMAMAC-protocol were compared for several metrics including energy efficiency, network lifetime based on first node death, and switch failure probability. The simulation modeling and analysis was done on the MiXiM simulator based on the OMNeT++ platform. Both variants have their own advantages and disadvantages and are applicable in different settings as explained in Chapter 2.

The contribution of the article "Model-based Specification and Validation of the DMAMAC Protocol" [52] was to describe the verification of the DMAMAC protocol via model checking. The article describes the modeling of both DMAMAC protocol variants Hybrid and TDMA using timed automata and the Uppaal tool [53]. The modeling approach and the models are discussed in detail along with initial validation using Message Sequence Charts (MSC). The representative network topology used for exhaustive verification queries is discussed along with the configurations used for verification and validation. Models created are validated also via queries and thoroughly verified using queries independent of configurations, configuration-specific queries, and real-time queries. The real-time queries had separate versions of the models created. The performance of the queries along with the system configuration used are listed for getting a better idea of the analysis. Further, the sequence diagrams from the Uppaal model simulation were used to validate the sequence diagrams from the simulation evaluation [51] and implementation evaluation of the DMAMAC protocol [54].

3.1 Simulation

Plethora of tools have been proposed over the years to perform wireless network simulation [62, 63]. In the recent years, many of the proposed network simulators have focussed particularly on Wireless Sensor Networks and Body Area Networks. OMNeT++ [64] is one of these simulator frameworks which performs better [65, 66] compared to other popular wireless simulators like ns-2 [67]. The OMNeT++ simulator has advantages over ns-2 in terms of customization and GUI features. OMNeT++ has also certain performance advantages [65] over ns-2. In this thesis, we used the MiXiM simulator based on the OMNeT++ platform to evaluate the performance of the DMAMAC protocol.

3.1.1 OMNeT++ MiXiM Simulator

OMNeT++ [64] is an open source simulation framework, that supports construction of simulators. The framework is modular and component based, where multiple submodules can be combined together to obtain a required solution, e.g. a network node with multiple protocols. The modularity allows modellers to replace existing modules with their own and analyze them for performance. The modularity also allows for re-use of existing modules and components. OMNeT++ is a discrete event simulator, thus the simulation procedure goes through a set of events occurring during the simulation timeline. Definition of a network over an OMNeT++ framework includes multiple types of files: Network Descriptor files (NED), INI or configuration files, the C++ files, message files, and XML files. The C++ files mainly describe the protocol modules under consideration. The NED files describe the physical characteristics of the nodes and the network, along with the C++ modules used by them. The XML files are for specifying configuration of the channel conditions, the topology and other configurational requirements. The message files are used to specify the packet types, and includes a detailed description of each field in the packets.

Over the years multiple network simulators have been built based on the OMNeT++ framework which now includes multiple C++ libraries supporting network simulations. MiXiM [68] and Castalia [69] are two network simulators used for WSN and Body Area Networks (BAN) respectively. OMNeT++ provides support for programming protocols spanning multiple layers to construct a virtual WSN with multiple nodes and simulate it over varying parameters. The modular architecture along with existing C++ libraries provided, lets users build their own protocols for performance analysis and testing.

In this thesis, we mainly refer to the MiXiM platform based on OMNeT++. MiXiM is a mixed simulator made with the combination of multiple simulation frameworks developed for mobile and wireless networks. It provides a rich C++ library with protocol models, and infrastructure required for a full network simulation. The MiXiM paltform consists of an Integrated Development Environment (IDE). The Eclipse-based simulation IDE provides a platform for development, simulation, debugging and analysis via a Graphical User Interface. The simulation support also provides support for running multiple simulations as batches, collecting required data, and analyzing this data via additional tools.

3.1.2 MiXiM model

The DMAMAC protocol model in MiXiM is created based on the specification of the protocol from [50]. A MiXiM node model is made up of multiple layers of protocol, each responsible for separate functions. MiXiM uses a modular approach, each of these layers constitute a simple module. Additionally, network descriptor (NED) files are used to describe a compound module made up of multiple simple modules. The compound modules can also be used to create a network based on the simple modules. The simple modules

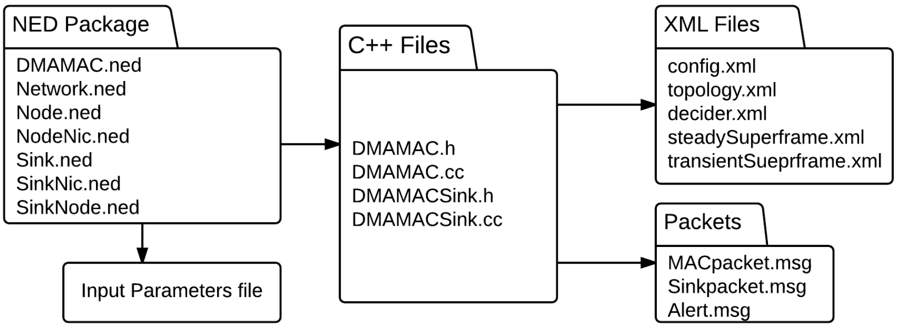


Figure 3.1.1: MiXiM DMAMAC model [51]

in this case include application, network, and radio modules connected to the DMAMAC protocol module.

The simulation architecture as shown in Fig. 3.1.1 is made of simple modules consisting of C++ files, network description with NED files, XML files for some user based input, packet types as message files, and input parameters file. Separate MAC modules are created for the nodes and the sink along with their own network descriptor files (NED files). Also, separate MAC modules are created for DMAMAC-Hybrid and DMAMAC-TDMA. Multiple configuration files are created, two of which are used to specify the superframe structure with one file for each superframe. The main packet types are: generic MAC packet, sink MAC packet, and alert packets. The configuration for execution is specified in the input parameters file. The configuration parameters also include the time duration for execution and the position of each node in the network. For application and radio layer modules, the existing modules from MiXiM were used.

3.1.3 Experiment Setup and Performance Analysis

The simulation setup consisted of 25 nodes at three levels in the tree topology excluding the sink. The nodes were comprised of 19 sensors and 6 actuators. We compared the DMAMAC-Hybrid and DMAMAC-TDMA with the GinMAC protocol. The comparison was done based on two important metrics: energy consumption and network lifetime in terms of first node death. The DMAMAC-TDMA protocol outperforms the other two protocols in most configurations. Also, we analyzed the state-switch failure possibilities in the DMAMAC-Hybrid variant, induced due to its alert message mechanism. The

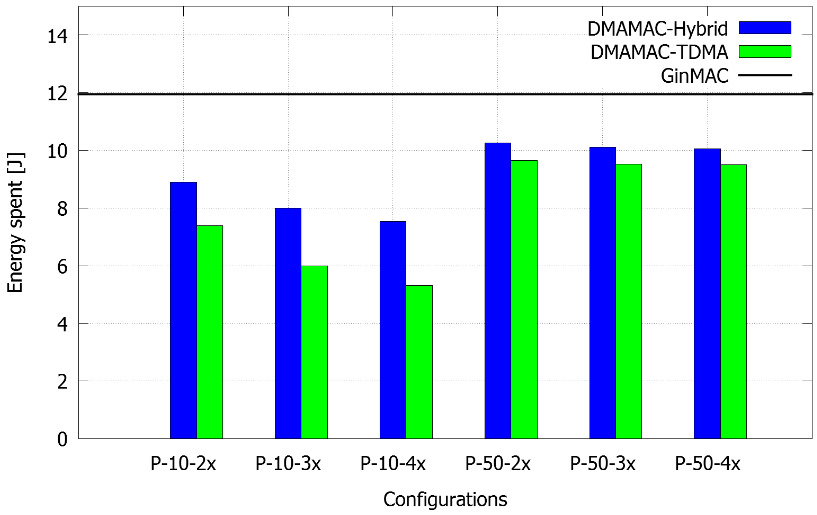


Figure 3.1.2: Energy consumption comparison in MiXiM [51]

DMAMAC-Hybrid variant performs decently in terms of the switch failure conditions.

The comparisons are mostly presented graphically, with additional tables for getting better insight. Among the results, we present the comparison of the two DMAMAC variants (TDMA and Hybrid) with GinMAC in Fig. 3.1.2 (a) and Fig. 3.1.3 (b). The GinMAC is presented as a black line as a benchmark and the DMAMAC variants with separate colours. Multiple configurations are used, mainly varying the ratio of transient and steady superframes in the execution, and varying the length of steady superframes. The "P-10-2x", is to be read such that 10 is the ratio of transient superframes in the entire execution (as 10% of the execution time) and 2x as steady superframe being twice the length of the transient superframe. All configurations were simulated using 100 repetitions each with varying execution sequences. Further, we briefly discussed the protocol performance for metrics including packet transmission delay, reliability and scalability.

3.2 Verification

The DMAMAC protocol is designed for process control applications with strict real-time requirements. The design of the DMAMAC protocol could benefit from formal verification to ensure behavioural correctness and con-

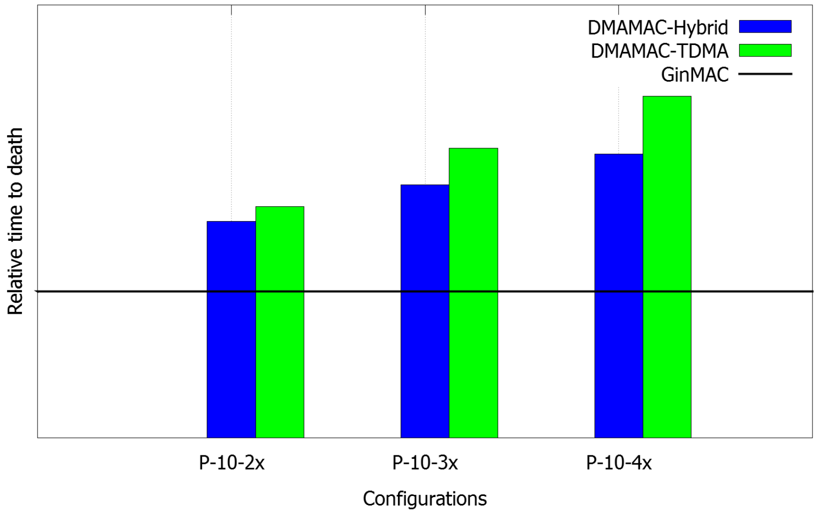


Figure 3.1.3: Node lifetime comparison in MiXiM [51]

formity to requirements. Existing formal verification tools like Uppaal [53], Prism [70] provide model-checking or model-based verification facilities based on exhaustive verification of a given input model. These verification tools consist of a verification engine equipped with query language that facilitate the verification process. The verification engine also provides counter examples to pin point faults in the design when a given query is not satisfied. In this thesis, we used timed automata and the Uppaal tool [53] to perform model-based verification of the DMAMAC protocol.

Uppaal has been previously used for verification of communication protocols [71, 72, 73]. The modeling and verification of the Lightweight Medium Access Control (LMAC) protocol is the most closely related work to the verification article [52]. The focus of the LMAC protocol modeling is to verify the slot selection and the collision detection module. For the DMAMAC protocol slot selection is done offline and no collision detection is included, also, the main focus is to verify the dual mode operation with switching. It requires a different model to represent the DMAMAC protocol than the ones used in the LMAC protocol verification article.

For the model-based verification of the DMAMAC protocol, we designed a generic MAC protocol model with DMAMAC protocol specific extensions. This extensibility feature of the designed Uppaal model is discussed later in this chapter. This generic design allows the re-use of the Uppaal model to obtain other Uppaal models of MAC protocols. The Uppaal model of the

DMAMAC protocol is also further used to validate the simulation and the implementation model of the DMAMAC protocol. This is done via sequence diagrams to verify specific functions/operations and is explained later in this chapter.

3.2.1 Uppaal Tool

Uppaal [53] is a formal verification tool providing features including modeling, simulation, verification, and performance analysis of real-time systems. The systems under consideration are designed as a network of timed automata [74]. The designed system is then validated and verified using simulation and query engine features provided by the tool suite. Each automata is used to define a process or an entity (e.g. a sensor node) and the network then makes the complete system. Each automata is defined using locations, transitions, and synchronization variables. The synchronization variables are used to synchronize multiple processes, each defined in it its own template. The programming syntax of Uppaal is C-like. Each template consists of declarations local to the template, and an automata descriptor. The network of timed automata is composed using a system declaration template. Uppaal has a graphical user interface to assist the users in creating, editing, simulating and verifying the input models. The description of the parts of the graphical user interface and its features are discussed below.

The graphical editor. Users can build a network of timed automata using the Uppaal graphical editor. It provides locations of different types, including: *initial, urgent, committed* and *regular.* Each editor template consists of a local declaration page where all the declarations and functions are defined.

The simulator. The Uppaal tool also provides a simulator in addition to the verification engine. Users can simulate their models step-by-step to perform state changes in detail. This enables the users to simulate particular scenarios to investigate the behaviour of the model.

Message sequence charts. Message Sequence Charts (MSC) can be obtained from the simulator, and can be used to validate the Uppaal model.

MSC provides a detailed step by step overview of the Uppaal model operation.

The verification engine. The verification engine takes query inputs to verify particular properties against model execution. The queries are defined in the query language based on a timed temporal logic [75] supported by Uppaal. Multiple queries in combination are used to validate and verify the model. Uppaal also provides the possibility to obtain simulation path for certain queries. For example, when a query that is expected to be satisfied is unsatisfied, the verification engine provides a counter example in the form a sequence of simulation steps.

Querry Language

For verification of the models in Uppaal the query language is used. The query language in Uppaal consists of state formulae and path formulae.

State formulae State formulae are logical statements with unary or binary operators used to check a particular state in the model execution. This is done using the model elements or data structures of the models. An example, is Sink.Collision, which specifies a boolean variable Collision in the process template of Sink. The query is satisfied if the Collision variable has the value true in a given state.

Path formulae Path formulae in general are used to verify properties like safety, liveness and reachability. They are used in combination with a state formula. The path formulae are used for exhaustive verification of all possible executions of the model, or in some cases to find one execution path where a query is satisfied.

The two path formulae used in Uppaal are:

- A[]ϕ (Always globally) - The state formula ϕ holds or is true in all states, along all paths.
- A<> ϕ (Always eventually) - For all execution paths ϕ eventually holds (in the starting state itself or in one of the future states).

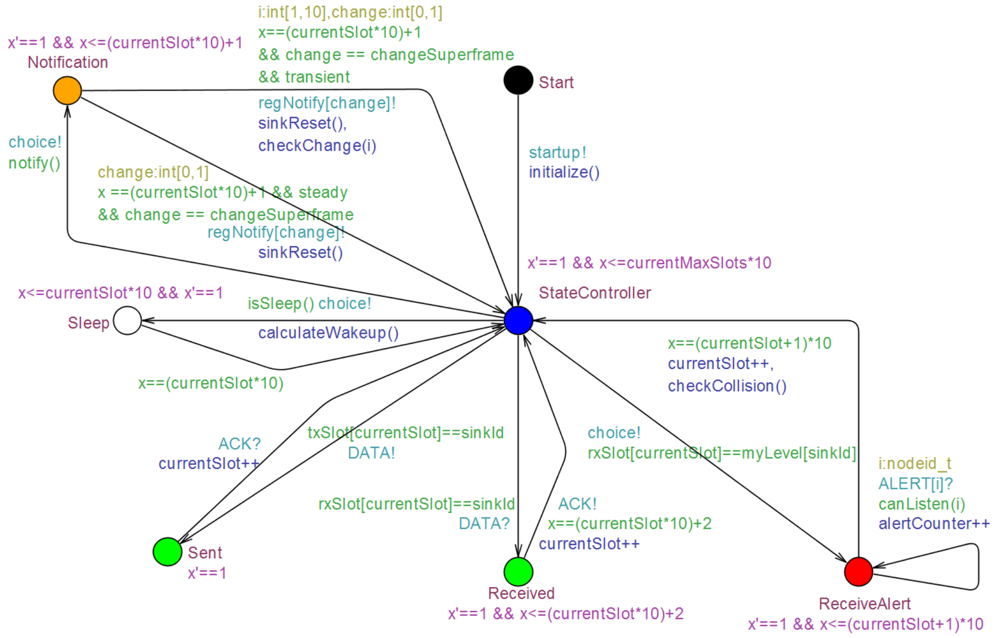


Figure 3.2.1: The DMAMAC sink model in Uppaal [52]

- E[]ϕ - There exists one execution path from the starting state where ϕ holds (or is true) in all states along at least one path.
- E<> ϕ - There exists one execution path from the starting state where ϕ eventually holds (or is true).

More details about modeling and verification in Uppaal can be found in [75].

3.2.2 The DMAMAC Uppaal Model

The DMAMAC Uppaal model consists of two IWSN elements, the sink and the node, each defined in a separate template. The Uppaal model of the sink operating on the DMAMAC protocol is shown in Fig. 3.2.1, and the node model is shown in Fig. 3.2.2. The Uppaal model is designed based on the protocol entity states in the DMAMAC protocol operation including: *sleep, data send/receive, notification send/receive*, and *alert send/receive*. Each state is represented as a location in Uppaal. The sink only receives alerts, thus the sink model does not have the alert send location. The nodes (sensor and actuators) only receives notification, and the sink sends them.

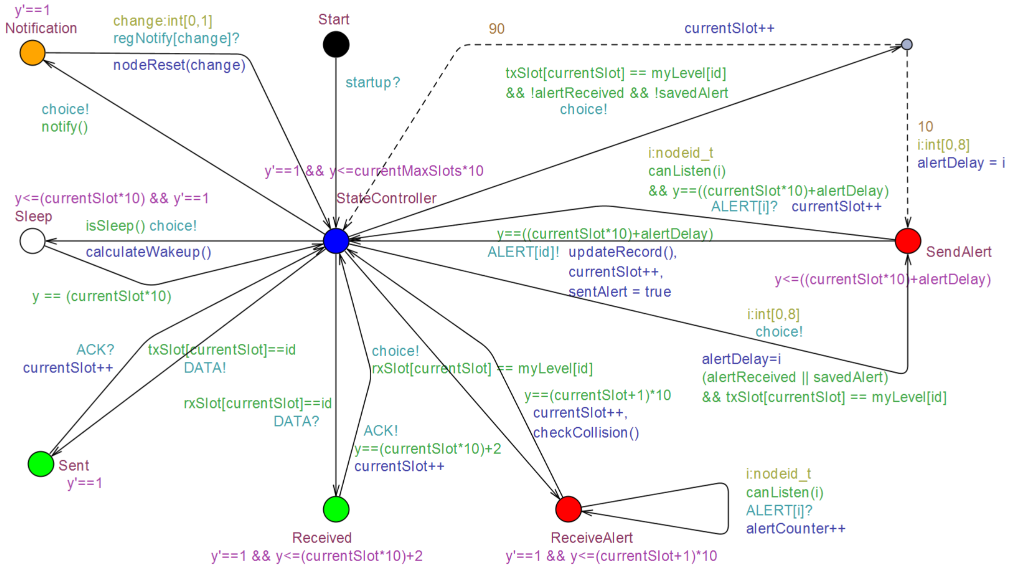


Figure 3.2.2: The DMAMAC node model in Uppaal [52]

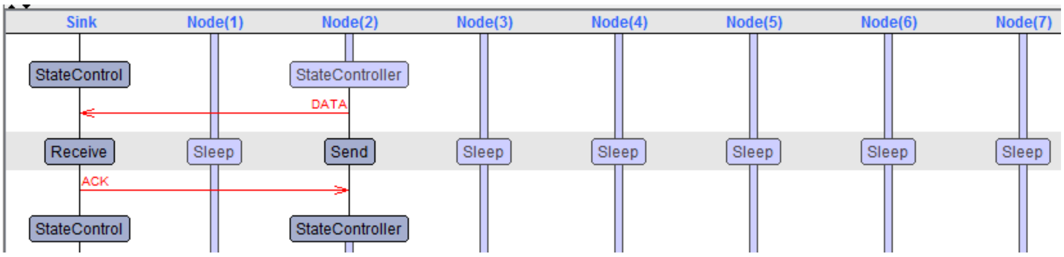


Figure 3.2.3: Message sequence chart for data transfer towards sink

3.2.3 Validation and Verification

We validated the Uppaal model of a network operating on the DMAMAC protocol using Message Sequence Charts (MSC). An example MSC for data transmission from a node towards the sink is shown in Fig. 3.2.3. Node 2 sends data towards the sink, and the sink acknowledges with an ACK message.

We also used the Uppaal query language and the verification engine to perform further validation. An example of validating the data transmission between the nodes, and between the nodes and the sink in the network was done using the following formulas. The formulae check for data transmission between all parent-child combination of sensor nodes in the network. *nodeid_t* is an identifier used to represent node numbers

Node data communication ∀ i,j ∈ nodeid_t:

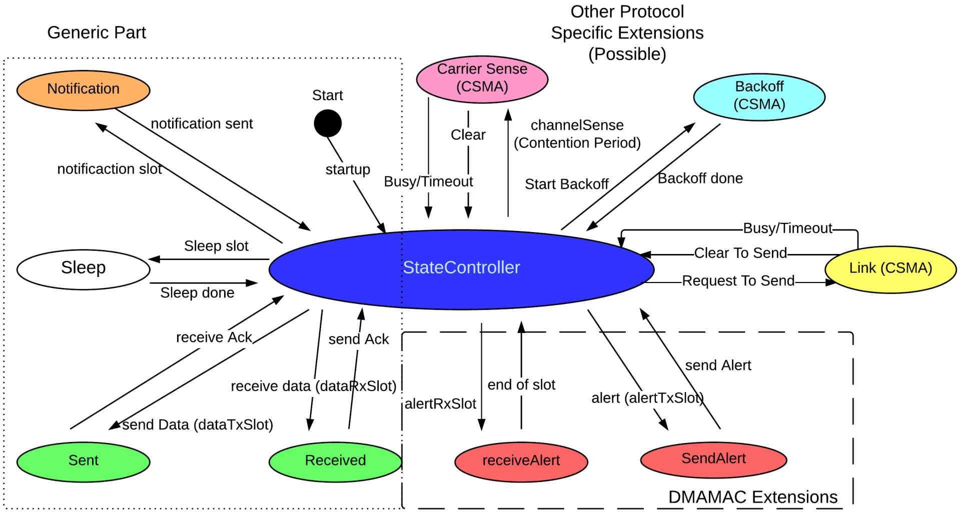


Figure 3.2.4: Generic FSM [52]

such that parent[j]==i: $A\Diamond$ (Node(i).Sent $\wedge$ Node(j).Received)

Sink data communication

$\forall$ i $\in$ nodeid_t such that parent[i]==sinkId: $A\Diamond$ (Node(i).Sent $\wedge$ Sink.Received)

Further, the DMAMAC protocol was also verified for functional and real-time properties. One of the important examples include the verification that the network is properly synchronized with respect to the operating modes. We verify that at all times both the nodes and the sink are in the same operational mode, i.e., in either transient or steady.

Consistent node mode $\forall$ i $\in$ nodeid_t:

$A\Box$ (Node(i).transient $==$ Sink.transient $||$ Node(i).steady $==$ Sink.steady)

These queries show selected examples from the article [52].

3.2.4 Extensibility

The finite state machine (FSM) based Uppaal model created for the DMAMAC protocol is designed in a generic MAC protocol context extended with protocol specific parts. The generic protocol operations included sending/receiving of data, ack and notification packets, and sleep operation. The generic FSM for MAC protocols and possible extensions are shown in Fig. 3.2.4. We have indicated the different parts: the generic part, the DMAMAC specific extension for alert message services, and other possible extensions. Some common MAC extended operations could include Carrier Sense Multiple Access (CSMA) operations. Some important CSMA operations included *carrier sense*, *backoff*, and *link*. *Carrier sense* is essentially used to make sure that the channel is clear. If the channel is busy the node performs *backoff* operation. In an event where the channel is clear, a link is established between the sender and the receiver using the *link* operation. Thus, the Uppaal model created for the DMAMAC protocol is re-usable and can be extended to model other MAC protocols.

3.2.5 Validation using Sequence Diagrams

The verification model created for Uppaal is used to also validate the simulation and implementation model (chapter 4). We present an example sequence diagram for data transmission in Uppaal (Fig. 3.2.5), which is used to validate the sequence diagrams generated for the same operation in the simulation and the implementation models. The detailed sequences could in general vary between the platforms, but the resulting behaviour is the same. Thus, we have validated the DMAMAC protocol operation across all platforms using the Uppaal sequence diagram as the base.

3.3 Contributions and Perspectives

The DMAMAC protocol in [51] was proposed as two variants: TDMA and Hybrid. The article presents a simulation on selected configurations and metrics. The assumption of ideal channel conditions is unrealistic, but does not affect the results of the simulation analysis. Although, we did study packet failure due to collision with respect to switch failure. The study of the effect of channel conditions with possible inclusion of Bit Error Rate (BER) could add into the existing analysis. BER is a channel configuration input to specify the error rate in received packets in terms of Bits. Similarly, it could add into the study of switch failures of the hybrid, and TDMA variant.

The article [52] presents an abstract timed Uppaal model of the DMAMAC protocol. Initially, the Hybrid variant of the DMAMAC protocol was

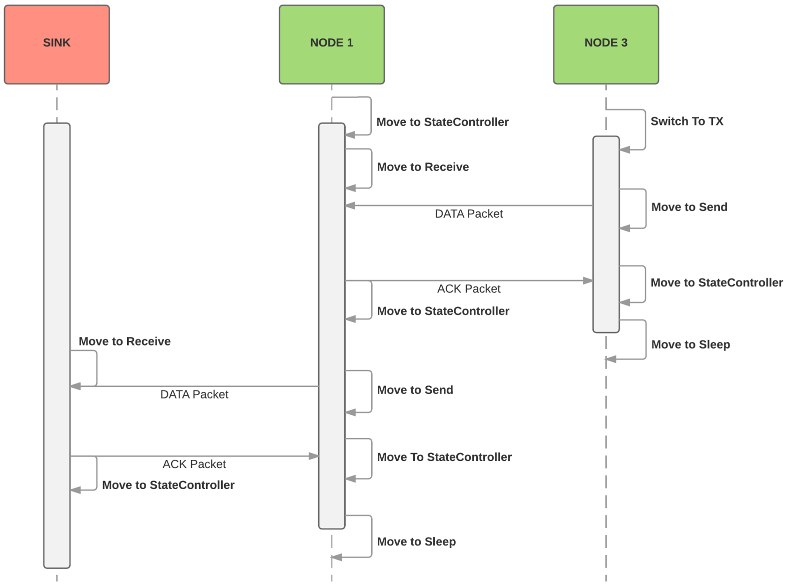


Figure 3.2.5: Data transmission sequence diagram in Uppaal [52]

created. Further, we created the TDMA variant for assessment. The current assessment of the DMAMAC protocol is qualitative. Uppaal further provides possibilities of quantitative and stochastic assessments [76], which has not been done as a part of the current work. Also, the modeling is based on assumptions and representative behaviour of collision and Carrier Sense Multiple Access (CSMA) operation.

Chapter 4

Implementation and Deployment

The last step in the design and development process is the implementation of the DMAMAC protocol on a hardware platform and deployment in a real world setting. We present an overview of this deployment and analysis phase for the DMAMAC protocol in this chapter based on the article [54].

The contribution of the article "Implementation and Deployment Evaluation of the DMAMAC Protocol for Wireless Sensor Actuator Networks" [54] is to provide the nesC implementation of the DMAMAC protocol and deployment experiences. We mainly focus on the goals set out for the implementation evaluation of the DMAMAC protocol. The DMAMAC nesC implementation model along with the used off the shelf TinyOS components are described to present the entire implementation. Further, the implementation code is deployed using the Zolertia hardware platform, and analyzed in a realistic industrial application scenario. Link quality assessment is conducted along with an observation of notification packet failures. The notification packets are responsible for performing the network wide switch ensuring that all nodes are performing in the same operating mode.

4.1 Background on TinyOS

TinyOS [77] is an event-based operating system developed to be used in resource constrained sensor/actuator hardware. The generic sensor network hardware is resource constrained in terms of processing power and memory usage. TinyOS uses the nesC programming language [78] for development of applications. The nesC programming model includes of *components*, which can be classified as *modules* and *configurations*. A *configuration* provides the details about how the components of the application/MAC is wired (connected) to other components. Components in general represent the software architecture. *Modules* provide the implementation related to an underlying component. Components provide services to other components, and also use services from other components. This is done via a set of *interfaces* defined as part of the signature of each component. This connection of interfaces across components is known as wiring. A given component *uses* one or more interfaces provided by other components included in TinyOS and *provides* interfaces through which other components can access the functions of the component under consideration.

Modules define the operations of the component. For the DMAMAC protocol this includes sending, receiving and forwarding packets, switch procedure, and network synchronization. *Commands*, *events*, and *tasks* are used in implementation of the components. *Commands* in the nesC programming language are used to define operations that can be used by external components or by other operations within the same component. *Events* are similar to interrupts and are handled by the module implementation of the component, e.g., reception of a packet either from the application/network or from the radio. *Tasks* are deferred procedure calls that run to completion.

4.2 DMAMAC Implementation Goals

The implementation step in the design and development process of the DMAMAC protocol was the final stage in validating its proper functioning. It includes deployment on a hardware with performance analysis in a real-world setting. The goals of implementation as mentioned in [54] are:

1. To validate the operation of the radio states via current consumption measurements.
2. To validate the implementability of the mode-switch procedure.
3. To study the impact of real conditions on the DMAMAC protocol operation, particularly the switch procedure of the protocol.

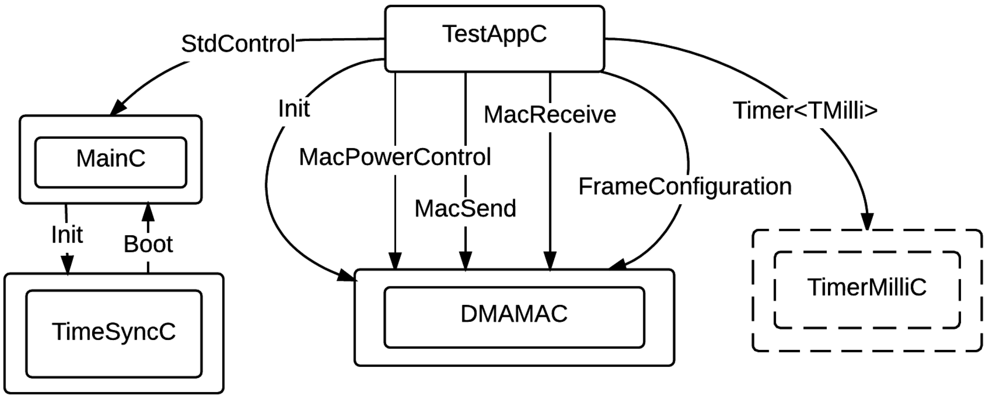


Figure 4.3.1: The nesC component diagram of the application to test DMAMAC

Radio state	Current measured
Sleep	0.5 mA – 0.6 mA
Tx	21 mA – 22 mA
Rx	23 mA – 23.5 mA

Table 4.2.1: Measured current in radio states.

A MAC protocol has three radio states of operation: Sleep, Transmit (TX) and Receive (RX). The energy efficiency is one of the important features of the DMAMAC protocol, which depends on sleep slots in operation. This is useful if the sleep state has a very low current consumption, we test the current consumption for each radio state to validate this. For measurement of the current and to validate the change of Radio states (Tx, Rx and Sleep) we used a small circuit. The measurements were made with a 3V power source placed along with a node and a 10 Ω resistor in series. The obtained values are shown in Tab. 4.2.1 and is in the expected range as shown in the data sheets for CC2420 [24].

The full version of the DMAMAC protocol was implemented along with packet transmission-reception, fully functional switch procedure, and quality assessment features. Thus, the second goal was fulfilled, and the implementation details follow.

4.3 The DMAMAC nesC Implementation

An application in TinyOS is described as a collection of interdependent components connected via their interfaces. An example TinyOS application used to implement and deploy the DMAMAC protocol is shown in Fig. 4.3.1.

The application used for deployment testing of the DMAMAC application is called TestAppC and consists of the configuration file of the application component. TestAppP is the underlying module implementation provided for the test application and is included in TestAppC. The test application mainly uses the interfaces provided by the DMAMAC component to provide a running application. In the current configuration the interfaces used in relation to the DMAMAC protocol component are:

- The Init interface to initialize the DMAMAC component.
- The MacPowerControl interface is used to switch the DMAMAC component ON or OFF.
- The MacSend and MacReceive interface is for sending application packets to, and receiving packets from the MAC layer (DMAMAC).
- The Frame Configuration interface is used to alter the length of steady and transient frames if required, directly from the application.

Other important components that the application is connected to are MainC, TimeSyncC, and TimerMilliC. MainC is a component that is the first one to be executed in a TinyOS application. It is a part of the application configuration by default. We also boot the TimeSyncC component used by the DMAMAC protocol for *time synchronization* operation, which is explained later in this chapter. The test application also uses a timer component TimerMilliC for sending packets at a given interval in order to emulate a typical sensor application. MilliC indicates that the unit of time is milliseconds. The implementation uses three packet types:

Notification The DMAMAC protocol *notification* packets contain 3 main data fields: *currentMode*, *sinkSlot*, and *change*. *currentMode* is used to convey the current mode of operation for the sink. *sinkSlot* provides the current sink slot information required for time synchronization (discussed below) which is essentially the slot number at which the sink sends the notification message. For changing the operating modes, the data field *change* is used. The notification packet used in the implementation is 7 bytes long.

Data The data packet is mainly designed for data transfer within the network. For implementation and analysis purposes, we use the data packet for collecting data from each sensor node. Important data fields are *notifyRssi*, *notfiyLqi*, *nbNotificationSlots*, and *nbNotificationPkts*. *notifyRssi* and *notfiyLqi* are used for assessing the link quality and deployment performance of notification packets. *notifyRssi* records the Received Signal Strength Indicator (RSSI) and *notfiyLqi* records the Link Quality Indicator (LQI) data for notification packets received from the sink. The RSSI and LQI data for each node is obtained using the TinyOS off the shelf component CC2420Packet. *nbNotificationSlots* and *nbNotificationPkts* are used to keep track of notification packet failure. The data packet is 17 bytes long.

Alert The alert packet is a short message to alert the sink about the change in process states. This alert results in change of operating modes. The switch from steady to transient is initiated by the nodes in the network. It contains only two fields *nodeId* and *alertSeqNo*, and is thus 4 bytes long.

We define constant numbers for the packet types. Apart from the packet types discussed previously, the operating modes *transient* and *steady* are implemented as integers for identifying the current mode in operation while switching between them.

4.3.1 Time Synchronization

Time Synchronization is a basic requirement of Time Division Multiple Access (TDMA) protocols like the DMAMAC protocol. Given the slot based execution procedure, every node should have the same notion of time. We use an off the shelf protocol, the Flooding Time Synchronization Protocol (FTSP) [79] as explained in [54]. In this overview, we further explain how the actual procedure is carried out for implementation and deployment on TinyOS. FTSP is provided as a part of the TinyOS protocol suite. In FTSP, initially we are required to designate a root node (sink node in our case). This root node serves as a global time reference to the rest of the nodes in the network. All other nodes in the network have their own notion of local time (LocalTime) and also the global time (GlobalTime) in the network. On startup, the GlobalTime and the LocalTime of any node is the same. Using the FTSP

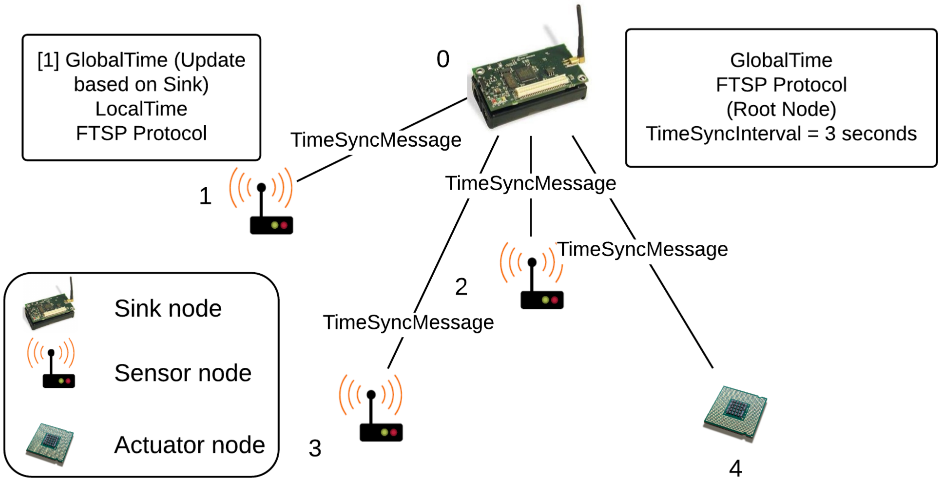


Figure 4.3.2: Global time synchronization process

messages obtained from the sink as shown in Fig. 4.3.2, the nodes update their global time. The sink sends FTSP update messages every n time units, based on a preset time synchronization interval (The sink as assumed is able to reach all nodes in the network). This is important to keep the nodes time synchronized over the duration of execution to overcome the known effect of clock drift on devices. Clock drift is an issue in the crystals used by the devices, which drifts out of synchronization from the rest of the network.

We execute the FTSP time synchronization protocol as a background process in the implementation of the DMAMAC protocol. The time synchronization algorithm is shown in the Pseudo code 4.1. Based on initial experiments with multiple nodes, we found that within the time the maximum number of table entries (8 for FTSP) in the local time synchronization table, the nodes get synchronized to the time of the sink. Thus, we use this as a condition to startup the scheduler in the node. In our implementation, we do this on the notification slot, where the sink node sends out its current slot information, which is used by the nodes to synchronize as they startup their local scheduler. Once synchronized, the nodes are in the same slot as the sink. Further, depending on the mode of operation of the network, the nodes switch between transient and steady mode on notification from the sink.

Listing 4.1: Time Synchronization Pseudo Code executed by all non-sink nodes

```
function Notification.Receive(NotificationPacket* notificationPkt)
    source <- notificationPkt.sourceId
    sinkSlot <- notificationPkt.slot
    //From time synchronization protocol
    numberOfTableEntries <- call TimeSyncInfo.getTableEnties()
    //Runs only once for startup, until scheduler is initialized
    if(source == sinkId
       && numberofTableEntries == maximumTableEntries
       && schedulerInitizlied == FALSE)
    then // Startup the scheduler
         Scheduler.start()
         // Synchronize to the sink's slot
         SchedulerControl.synchronize(sinkSlot);
         // Scheduler Initialized
    endif
end function
```

4.4 Deployment Setup

The network setup used for implementation and deployment is shown in Fig. 4.4.1 [54]. We use zolertia Z1 [80] motes for the setup. The zolertia nodes used for the experiment are powered by MSP430F2617 low power micro-controller with 16-bit RISC CPU at 16 MHz clock speed. It includes 8KB RAM and 92KB flash memory, and has an IEEE compliant CC2420 transceiver. The transceiver operates at 2.4GHz with a data rate of 250Kbps. The regular Z1 motes used for sensor and actuator devices have an integrated ceramic antenna. The sink on the other hand is based on the Z1 starter platform mote with an external +5dBm antenna to have a larger range than other nodes. The motes can be powered by either batteries or via a Universal Serial Bus (USB) power. We use the tree topology with the sink as a wire powered device, and the rest of the motes as battery powered devices. The battery powered devices are powered by 2 AA batteries. In the notification slot, the sink sends out the notification packet to the entire network. With the increased range of the +5dBm antenna, the sink is able to reach all nodes in

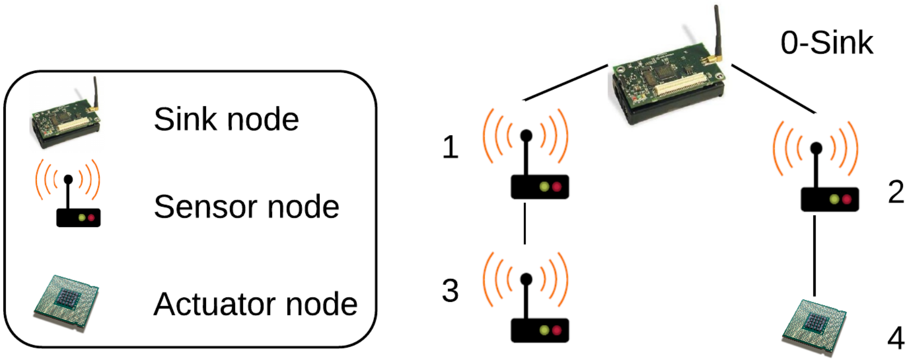


Figure 4.4.1: The network setup for deployment test

the network in one hop.

For our experiment, we collect the data from the motes using the serial communication port of the sink. Each mote sends out a packet with collected data relating to the previously received notification packet. Either directly to the sink (in case of node 1 and node 2) or via an intermediate hop (in case of node 3 and node 4). The packets enclose RSSI and LQI data for the notification packets received from the sink, and the number of notification packets that failed i.e., packets that were not received in the considered notification slot. The RSSI data presents a link quality picture based on the received signal strength for each notification packet received. The LQI indicator records link quality based on correctly received packets, for errors in received packets lower LQI value is recorded. This data collected in the sink is stored in a text file, and is used to create a graphical representation of the collected data using python scripts and the gnuplot tool.

4.5 Deployment Analysis

We performed four experiments using a combination of two particular scenarios. All the experiments were conducted in a lab shown in Fig 4.5.1. The scenarios were aimed at getting a trial run closer to what an industrial application would look like. To emulate an industry like environment with operational appliances, we used multiple high wattage (3kW) asynchronous motors operating at a voltage of 325 V. This is essential to test if any interference is created in the operation of the deployed IWSN due to these appliances. The two scenarios are: with motors on, and with motors off. Further, we also var-

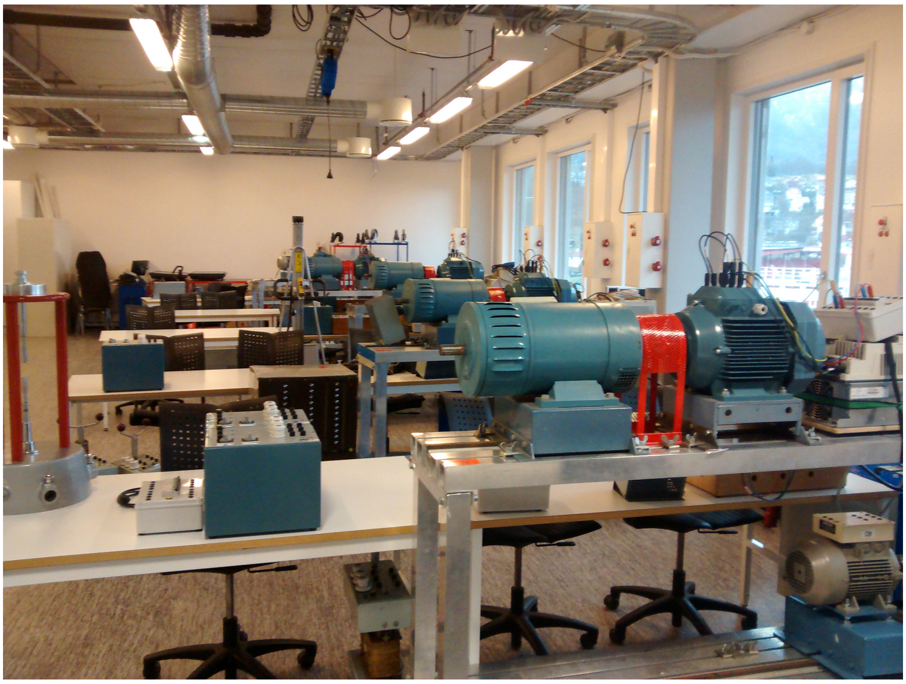


Figure 4.5.1: The lab environment for experimentation

ied the position of the wireless sensor nodes, particularly with line of sight placements and non-line of sight placements. In line of sight placements, all nodes are placed in direct line of sight of the nodes they communicate directly with. All these scenarios, including the considered metrics for link quality analysis are presented in [54]. Additionally, we collected notification packet failure statistics. Both, the link quality analysis and packet failure analysis were aimed at notification packets. The third goal for the implementation deals with switch procedure for the operating modes of the DMAMAC protocol. The notification packet is responsible for implementing the network wide switch which is thus critical to the switch procedure.

The results obtained in terms of RSSI and LQI for one of the non line of sight placement scenario is shown in Fig. 4.5.2. The RSSI results in particular indicate large variation in received signal strength over the duration of the experiment indicating some effect on it. The LQI results on the other hand, are within a limited interval during the entire duration, with some outliers. This indicates the link quality was not affected severely from the variations in the RSSI. The LQI and RSSI in combination give a picture of the link quality. Further, we recorded notification packet failure statistics. Each node reports the notification packets not received in its notification slots. Since

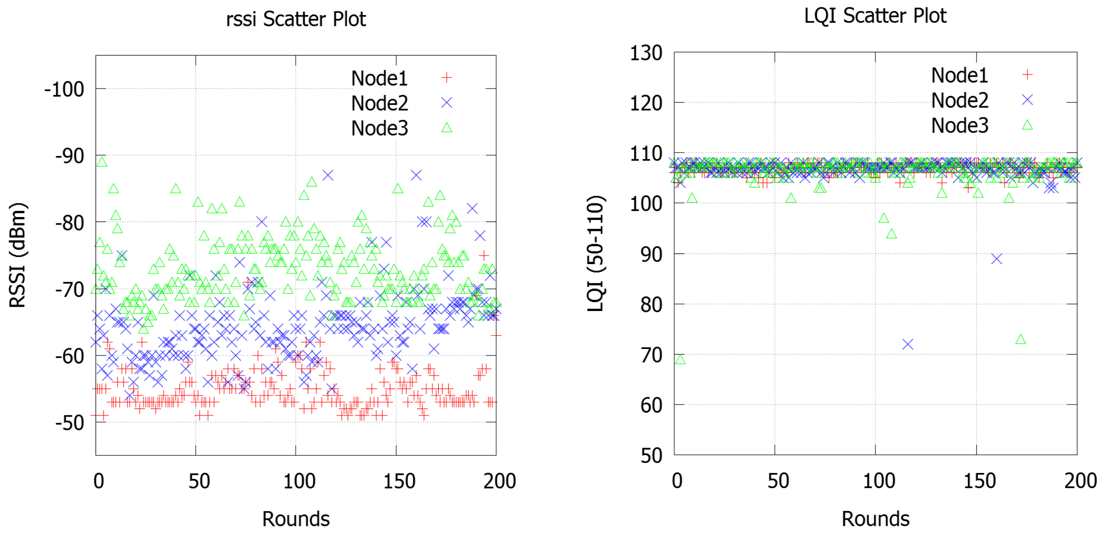


Figure 4.5.2: Experimental results for motor room scenario - arbitrary node placement. [54]

Scenario	Open room non-LOS	Open room LOS	Motor room non-LOS	Motor room LOS
Node 1	0/300	0/300	1/300	3/300
Node 2	1/300	1/300	3/300	3/300
Node 3	6/300	2/300	6/300	4/300

Table 4.5.1: Notification packet failures on nodes 1-3 line of sight (LOS) and non-line of sight (non-LOS). [54]

notification is a broadcast packet, no acknowledgement is used and the failure is tracked at the nodes. The failure statistics for the sensor nodes is presented in Table 4.5.1. The results are presented as number of notification packets failed in 300 notification slots during the experiment. For initial assessment, this indicates a low failure, which as a consequence of a limited effect on the performance of the DMAMAC protocol. The results for the other scenarios can be found in the article [54].

4.6 Contributions and Perspectives

An implementation of the DMAMAC protocol was created using the nesC programming language developed for TinyOS. The implementation was tested for change of radio states via current consumption measurements. The switch procedure was verified to operate correctly as per requirement. Finally, we conducted experiments to analyze link quality in varying channel conditions with nodes operating on the DMAMAC protocol.

The DMAMAC protocol implemented in TinyOS can also be configured to operate in co-operation [81] with IWSN operating on other operating systems, ContikiOS [82] is an example. Also, the DMAMAC protocol implementation can be ported easily to ContikiOS. TinyOS and ContikiOS have similarities in the programming paradigm, which can be exploited to easily port TinyOS code into ContikiOS code. This allows for the testing of the DMAMAC protocol in other hardware platforms. CC2560 [83] is an example hardware that supports ContikiOS implementation.

The article [54] is an initial investigation of the deployment performance of the DMAMAC protocol. We mostly present preliminary results in varying configurations and scenarios. The packet loss and link quality assessment could be further consolidated with extensive experimentation for longer duration. Currently, based on the goals, a small topology is used for testing. A larger topology with higher number of nodes could present a new perspective to the protocol operation. The testing of the protocol in a real process control setting would give a further idea of its operating performance. Also, the inter-operation of the protocol within the existing framework of wireless standards could be an interesting investigation, since in these experiments we tested it only as standalone.

Chapter 5

Towards A Model Based Approach for MAC Protocol Development

As presented in earlier chapters, the DMAMAC protocol is designed using the traditional approach. We initially identified the application domain as the process control domain. After which, the requirements for a Medium Access Control (MAC) protocol (narrowed focus) is drawn out based on the requirements of applications in the process control domain. Mainly, the dual process states result in the two operating modes of the DMAMAC protocol [50]. The DMAMAC protocol design [50] is based on the design of the GinMAC [44] protocol having similar requirements. The designed protocol was implemented in the network simulator MiXiM for performance analysis and verification of requirements. This step was discussed in [51] and in Chapter 3. The DMAMAC protocol was also verified for consistency in design and behaviour using formal verification tool Uppaal [53], as presented in [52] and in Chapter 3. The protocol was implemented using nesC on the TinyOS platform, and deployed using the hardware platform Zolertia Z1 [80]. The implementation was used to perform link quality analysis and validate the working of the DMAMAC protocol. The implementation step is discussed in [54] and Chapter 4.

Given the traditional approach taken in the design and development of the DMAMAC protocol, we investigate alternative methods that could improve the design and development. The Model-Driven Software Engineering (MDSE) approach proposed in [55] is a step further in this investigation, which was further extended in [57]. In this chapter, we discuss this alternative development approach by providing an overview of the articles [55, 57]:

The contribution of the article "Towards a model-based development approach for wireless sensor-actuator network protocols" is to present the initial idea of an MDSE approach for MAC protocols. The article outlines the MDSE approach based software code generation architecture. The entire process mainly uses: a modeling tool, the PetriCode tool for code generation based on platform specific code templates, and platform specific tools for evaluation of the generated MAC protocol artifacts. The position paper was extended in the article "Model-based Development for MAC Protocols in Industrial Wireless Sensor Networks" [55]. We present the completed MDSE approach for MAC protocol design in CPN tools, discussing the PetriCode templates for simulation and deployment platform. The simulation platform source code generated is simulated on the MiXiM platform and analyzed for performance of the protocol. The implemented code for deployment is deployed using a hardware platform and tested for link quality. We use the GinMAC protocol as the running example to present the MDSE approach. Specific parts of the model, the slotter function in particular is presented as a CPN model. The code templates for conversion of this model to MiXiM and TinyOS code are also presented.

5.1 Background on MDSE

Model-Driven Software Engineering (MDSE) [56] is an approach in the software development and engineering domain. The MDSE approach is used to produce high-quality software with higher confidence and trust in the inherent design. This is achieved using model-driven techniques. MDSE is an approach which uses an architecture similar to the Model-Driven Architecture (MDA) [84, 85] proposed by the Object Management Group (OMG). MDA is mainly based on Unified Modeling Language (UML) models and code generation from UML models. In our MDSE approach, we use alternative modeling languages as a starting point. Advantages of an MDSE approach are similar to the MDA advantages [85] and include:

Productivity The shift of focus from the development of Platform Specific Models (PSMs) to Platform Independent Models (PIMs) results in productivity advantages. Firstly, PIMs are abstract models and less complex than the PSMs, thus less coding/development effort is required. Secondly, there is a reduction in time from design to implementation with automatic code generation in place to transform PIM to PSM.

Portability PSMs are created using code templates based on design patterns. With the use and focus on PIMs, additional PSMs can be obtained from a single PIM using a different set of code templates. Developers can also re-use existing code generation tools for popular platforms, thereby reducing development time for the implementation (from PIM to PSM).

Flexibility Easier to modify requirements, if and when required. After the performance analysis phase, the PIM can be modified to change the design, re-run code generation and obtain the required performance. Further, PSMs are generated using automated tool chains.

The MDSE approach begins with an abstract and a platform independent representation of the software to be developed, MAC protocol software in our example. The abstract modeling focuses on modeling of the behaviour of the protocol. The abstract models are validated against the requirements specification and verified against the behavioural requirements. Further, these abstract platform independent models are converted into implementation code for multiple platforms serving different purposes. The primary purposes in our case consists of: software verification, simulation-based analysis, and deployment based analysis. The deployment based analysis can also be performed using multiple platforms to ensure the applicability of the software across the domain of application. The application of the MDSE approach has been described in [86, 87, 88, 89, 90, 91, 92]. Perspectives on related work are presented later in the chapter.

Traditionally, in the IWSN domain, the development of protocols goes through a sequence of steps, similar to the steps in the thesis, also shown in Fig. 5.1.1:

Requirement specification Based on the application (in our case process control) we capture the requirements, mainly operational performance requirements.

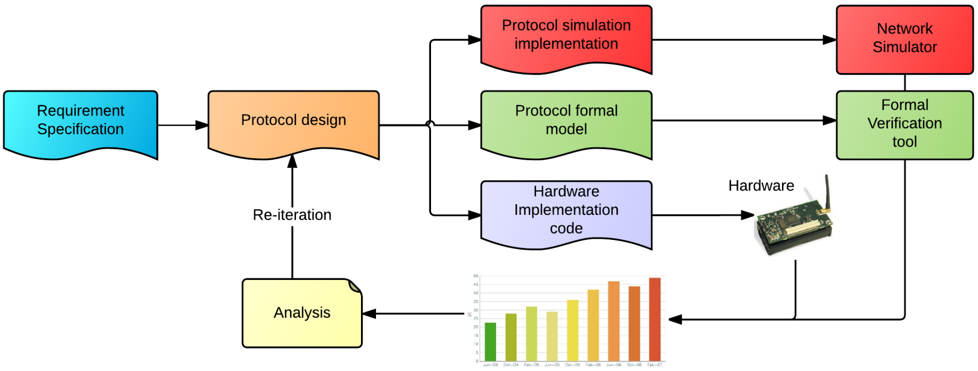


Figure 5.1.1: Development framework used for the DMAMAC protocol

Design Based on the requirements collected the DMAMAC protocol was proposed, and presented in simple language as set of statements. Later, a design in terms of a superframe schedule was designed.

Analysis The designed protocol is analyzed for completeness, correctness, and performance. In this thesis, important performance requirements are: energy efficiency and real-time requirements (low-delay, robustness). We analyzed the DMAMAC protocol using a network simulator for performance, and using a model-checking tool for correctness and completeness. Alternatively, the protocol design can be mathematically analyzed similar to [43, 44, 46, 61, 93, 94, 95]. Other protocols considering the simulation analysis are [47, 61, 96]. We have also performed an initial analytical evaluation of the DMAMAC protocol in the article [50].

Implementation The final step in the development of the protocols is implementation, where protocols are implemented on a hardware platform and tested in real-world environment. Some examples are [43, 44, 45, 47, 61, 95, 96]. The implementation step lets the developers analyze the real-world performance of the protocols, where the channel conditions can vary.

Each of these steps uses a separate representation of the considered protocol, the DMAMAC protocol in our case. Also, all these representations (or models) are converted manually from one representation to another. The

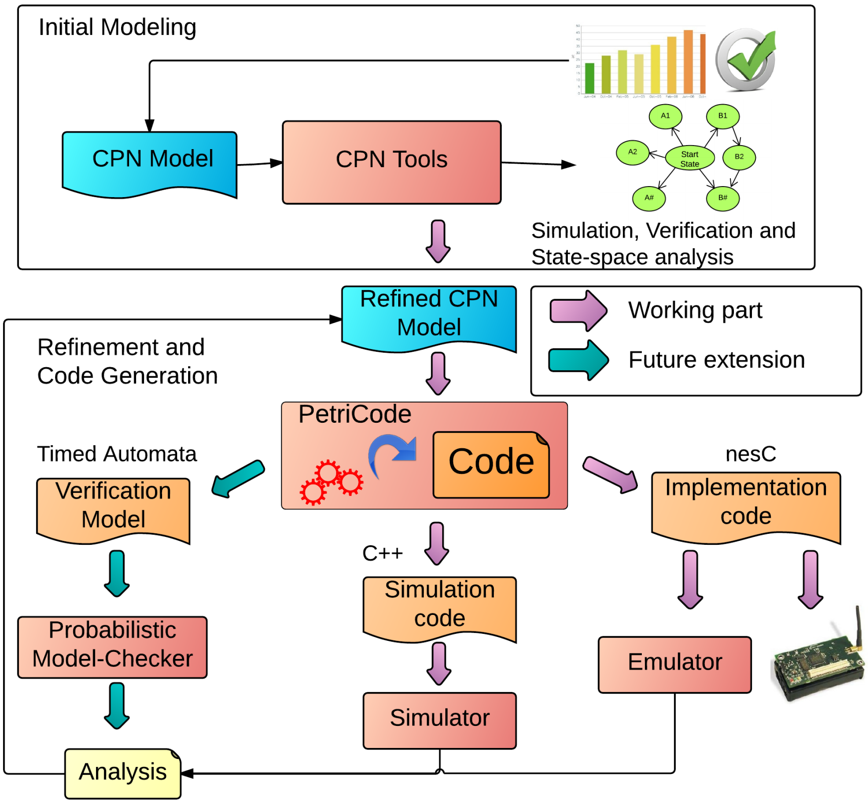


Figure 5.2.1: The proposed MDSE approach architecture [57]

conversions as shown in the design framework in Fig. 5.1.1 from protocol design to multiple PSMs for network simulators, formal verification tools, and hardware platforms are made manually. The manual conversion is generally error-prone and time consuming. To overcome this, in the software engineering domain Model-Driven Software Engineering approach is used along with tools to assist with automatic/semi-automatic code generation.

5.2 An MDSE Approach for MAC Protocols

The proposed code generation approach for the MAC protocols for IWSN was initially published in [55] and extended in [57]. The working part of the proposed MDSE approach software architecture is shown in Fig. 5.2.1. For the abstract representation we use CPN tools, based on the Coloured Petri Nets (CPN). The initial CPN model created in CPN tools is analysed for performance using simulation and behaviour using state-space analysis [97]. Using the obtained analysis, the CPN model is refined in multiple steps. The refined CPN model is the input the code generation tool PetriCode [49], discussed in the next subsection. We obtained multi-platform code generation from the CPN model to the platforms MiXiM and TinyOS. MiXiM [68] is a wireless network simulator platform. TinyOS [77] is an event-driven operating system for resource constrained hardware. Further, analysis of the generated protocol performance (GinMAC) was performed in the integrated tools of MiXiM, and on the Zolertia Z1 [80] platform with TinyOS implementation code. The MDSE approach can be extended to perform further formal verification. Example tools for formal verification of real-time systems like IWSN nodes are Uppaal [53] and PRISM [70]. These tools allow for thorough verification of protocol behaviour, also of real-time behaviour and probabilistic behaviour.

5.2.1 PetriCode

The PetriCode tool [49] is used for the code generation step in the MDSE approach. The tool is built upon the following concepts. *Pragmatics* are the annotations used to identify the model artifacts important for the code generation. *PA-CPN*. Pragmatic Annotated Coloured Petri Nets (PA-CPN), is a subclass of CPN models used for the code generation process. The PA-CPN model is platform independent and corresponds to PIM in the MDSE approach. Abstract Tree Template (ATT), is a part of the code generation process. The PA-CPN is first converted to an ATT before the final code generation. *Code Generation Templates* are template definitions provided to assist in the code generation. Each pragmatic is associated with a template definition for each platform (TinyOS and MiXiM in this case). In our example, each pragmatic has two code generation templates associated with it, one for MiXiM and one for TinyOS. The pragmatics are converted to a given platform code using the code generation templates. The *Template bindings* is the map, which maps the pragmatics to the corresponding code generation template.

The PetriCode architecture is shown in Fig. 5.2.2. The PetriCode tool already included templates for Java and Clojure as shown in the Fig. 5.2.2 for communication protocols. Each platform defines a bindings file and a set of code generation templates, one for each related pragmatic included in

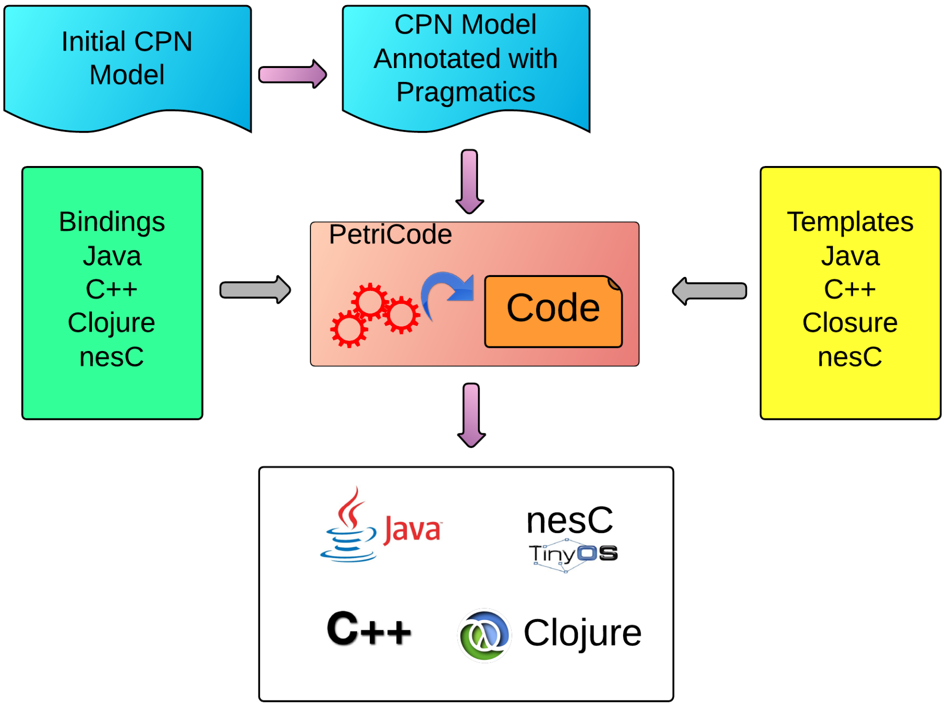


Figure 5.2.2: PetriCode MDSE approach

the CPN model. The PetriCode tool, based on the PA-CPN input creates an ATT. Further, using the template bindings and the code generation template, the resultant code is generated. An abstract PA-CPN model in PetriCode is designed hierarchically [49]. Three important levels in the hierarchy include:

Protocol System Level The top level module of the PA-CPN model which contains all the essential elements for the protocol execution, along with a channel (wireless channel in this case). It mainly describes each principal element in the PA-CPN model and their connection to a channel. An example for principal element is a MAC protocol.

Principal Level Each principal element consists of a collection of services that make up the principal level. The collection of services are typically the functions provided by the MAC protocol.

Service Level Each service is detailed in its own separate modeling file, constituting the service level. An example function is packet transmission function.

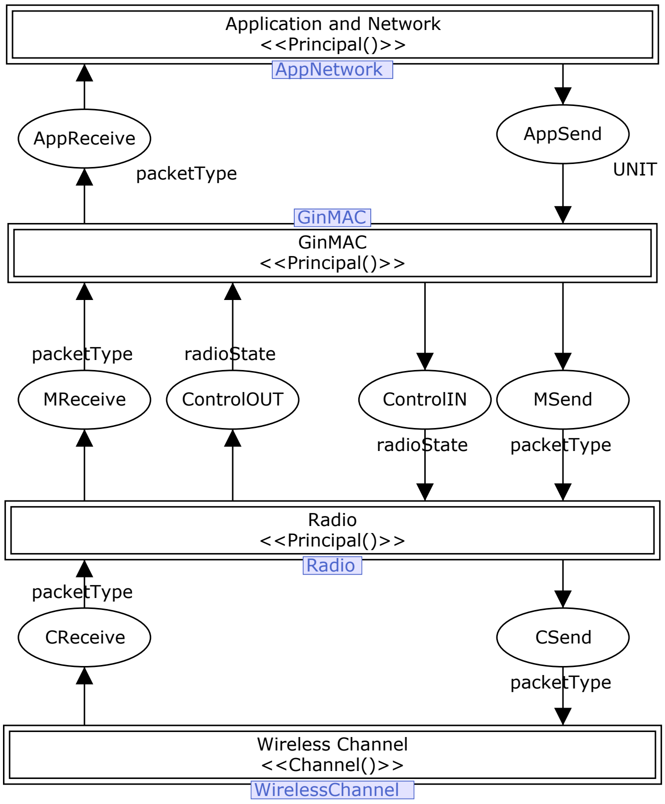


Figure 5.3.1: The Protocol System level of the GinMAC CPN model

5.3 GinMAC CPN model

An example for the protocol system level using the GinMAC PA-CPN model is presented in Fig. 5.3.1. The Protocol System level consists of other functions/layers required for a complete sensor/actuator execution, along with the wireless channel. The application and radio modules are required for the complete setup. The CPN model of the GinMAC layer in the principal level is shown in Fig. 5.3.2. The principal level represents services in the GinMAC protocol. For example, HandleRadioPacket, HandleUpperPacket, for handling packets from radio module and the application module respectively. The Slotter function is for the Time Division Multiple Access implementation, to cycle through each slot. The HandleRadioControl function is used to handle control messages received from the radio module. The Sender function handles the sending of the packets with appropriate markings (data,ack,notification). The service level model for the Sender function is shown in Fig. 5.3.3.

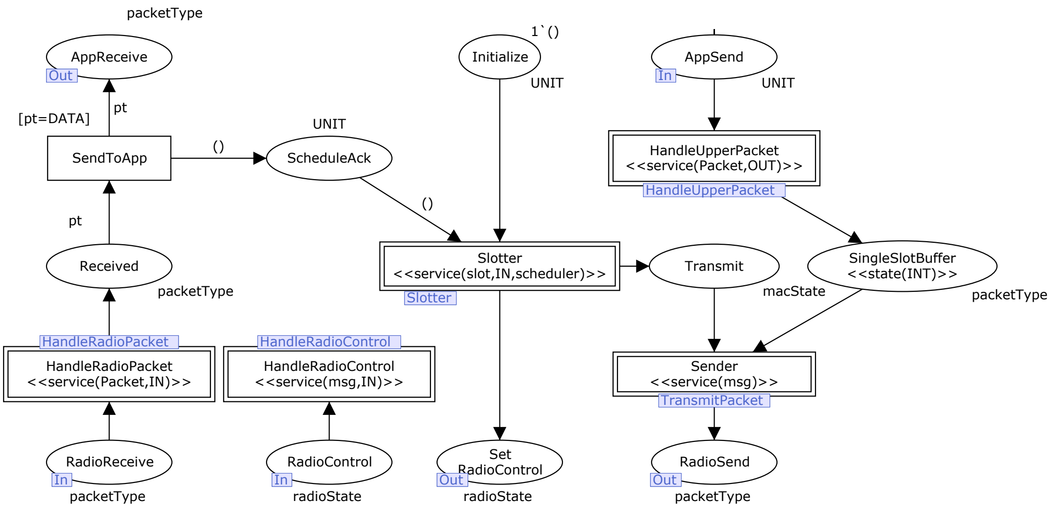


Figure 5.3.2: The Principal level of the GinMAC CPN model

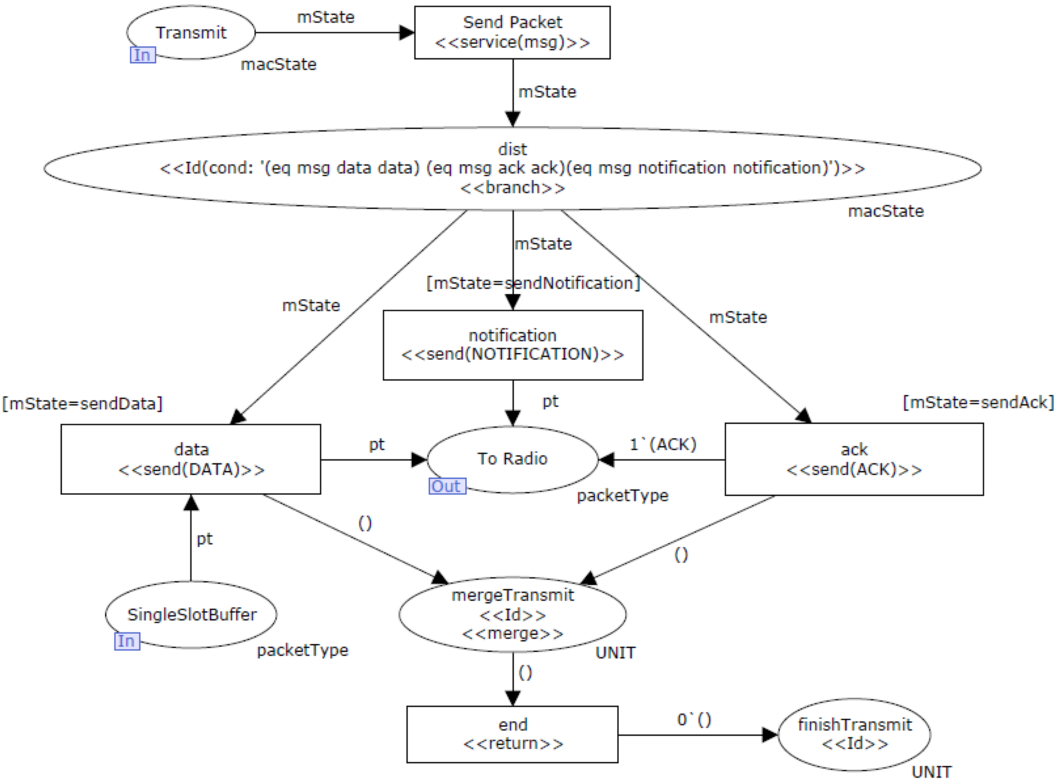


Figure 5.3.3: Service level example for packet transmission in the GinMAC protocol

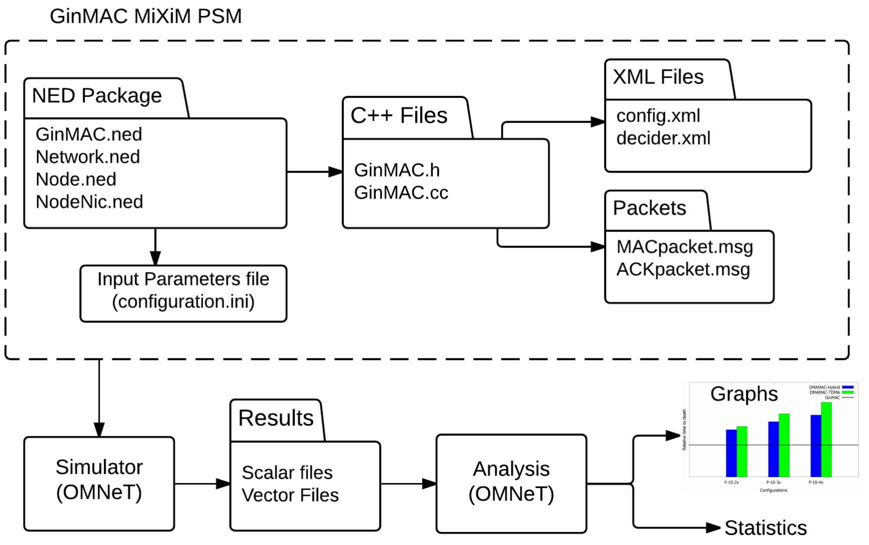


Figure 5.4.1: PSM and simulation flow for GinMAC in MiXiM [57]

5.4 Platform Specific Models

In the presented articles [55, 57], we describe the MDSE approach along with a created platform independent PA-CPN model of the GinMAC protocol, the running example for the MDSE approach. We have added in the template bindings and code generation templates for the MiXiM platform as C++, and TinyOS platform as nesC. Essentially, a common abstract model is mapped to two separate platforms with separate execution models. The obtained implementation code and their analysis is discussed below.

5.4.1 MiXiM

A PSM for MiXiM [68], a wireless network simulator platform is obtained using code the code generation templates added to PetriCode. The obtained PSM model, along with the simulation flow is shown in Fig. 5.4.1. The obtained MiXiM PSM is simulated on the MiXiM OMNeT platform for a specific duration of time and statistics are collected during the simulation. Using the analysis support provided by the MiXiM tool, the collected statistics are analyzed via graphical representations. A specific network topology is used for the simulation. We recorded packet transmission and reception

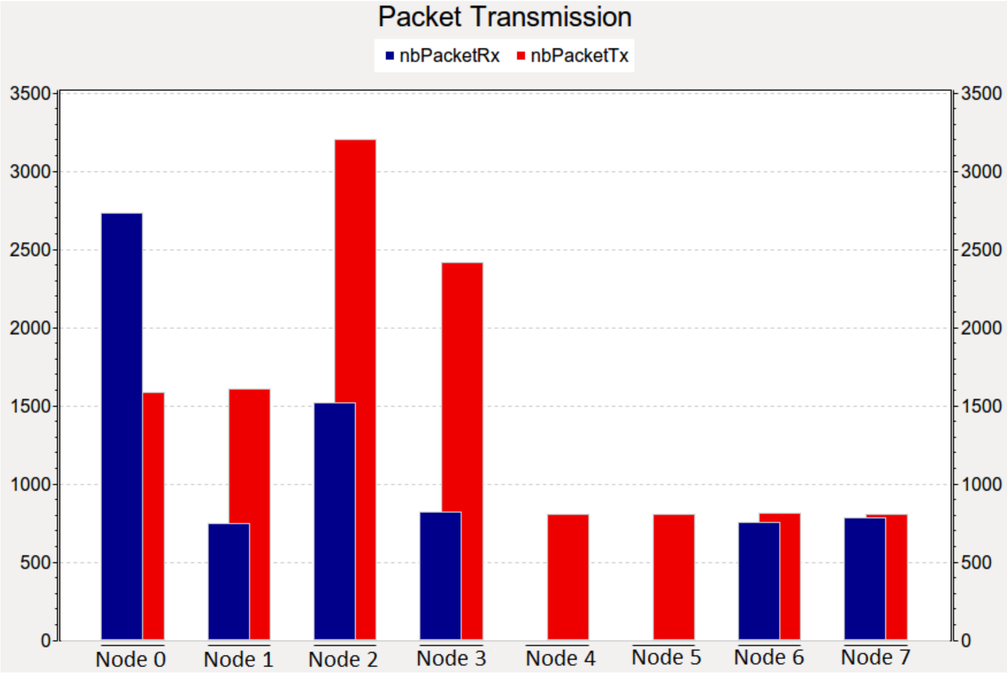


Figure 5.4.2: The packet reception and transmission statistics for the GinMAC protocol

statistics and network lifetime based on first node death. Graphs representing the recorded data are shown in Fig. 5.4.2 and Fig. 5.4.3. The packet reception and transmission graph shows number of packets transmitted and received by each node during the simulation. Node 4 and Node 5 are two leaf sensor nodes, thus they do not receive data. The network lifetime shows the death of node 2 to be the first among the nodes in the network. Further, the analysis feature of the OMNeT++ MiXiM platform can be used to analyze the protocol based on other metrics.

5.4.2 TinyOS

Similar to the MiXiM PSM generation, a PSM for the TinyOS platform is generated. Both PSMs are generated from the same input PA-CPN model of the GinMAC protocol. The generated implementation code in nesC for TinyOS is deployed on a hardware platform, Zolertia Z1 [80]. We conducted experiments based on a network topology. The results were logged on a computer and analyzed using generated graphical representations. For the hardware implementation we obtain link quality measurements. Based on the values provided by TinyOS off the shelf components, we obtain Link Quality

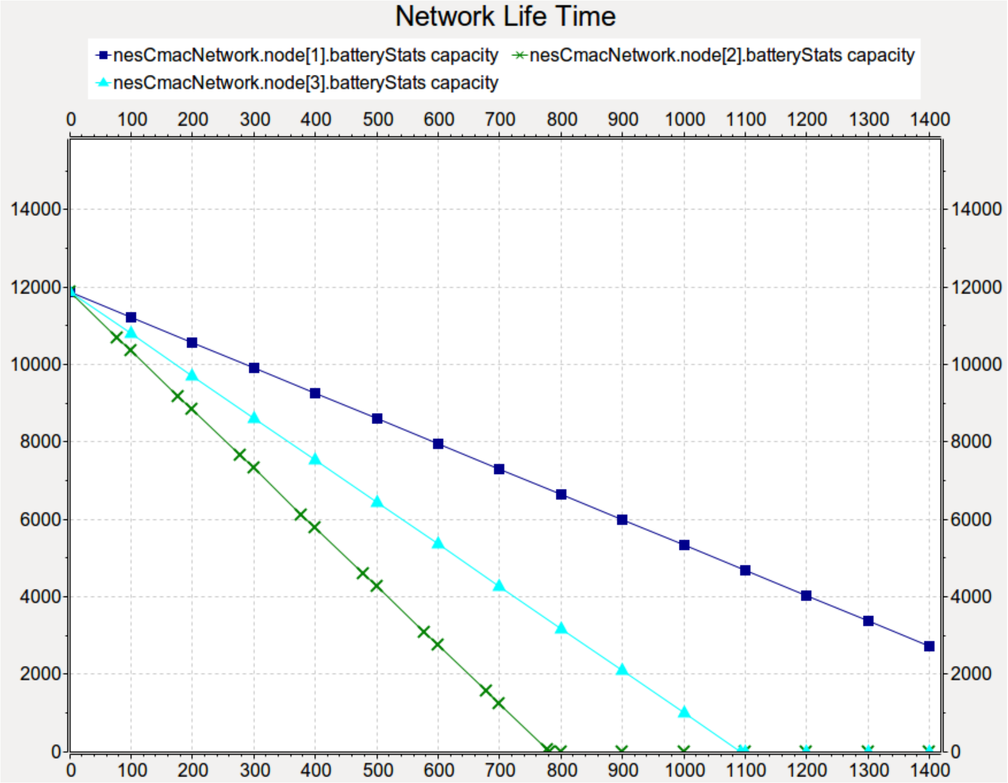


Figure 5.4.3: Network life time in terms node deaths ([57]

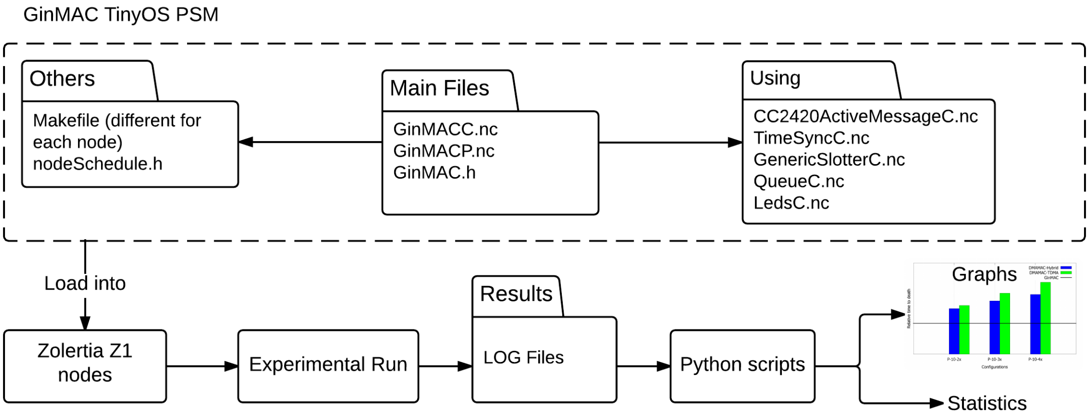


Figure 5.4.4: The PSM and implementation-deployment flow for GinMAC in MiXiM [57]

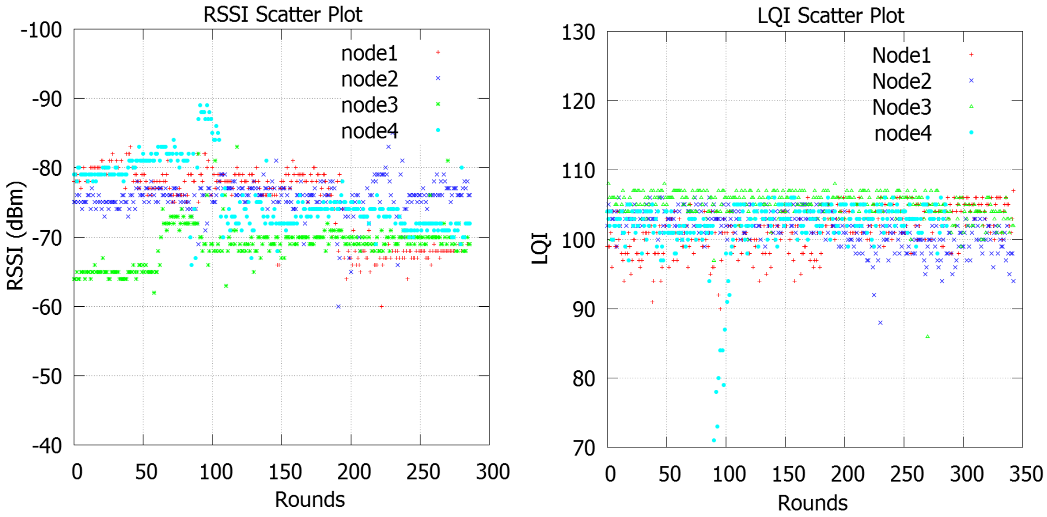


Figure 5.4.5: Link quality based on RSSI (a) and LQI (b)[57]

Indicator (LQI) and Received Signal Strength Indicator (RSSI). The LQI and RSSI collectively gives a picture of the link quality between the nodes in the network. For reading, LQI is measured between 50 to 110 with 110 being the best measure, and RSSI is measured between -45 dBm to -95 dBm with measures towards -45 indicating good quality. The measured values for the deployment test done in the electrical lab facility is shown in Fig. 5.4.5.

5.5 Contributions and Perspectives

The chapter summarizes alternative development methodology to develop MAC protocols. The main aim of this chapter is to give a detailed motivation for the proposed MDSE approach, in terms of the design and development of the DMAMAC protocol.

Model-based development approaches have been previously applied in WSN domain [86, 87, 88, 89, 90, 91, 92, 98]. The approaches focus on rapid prototyping of a development model and are based on either domain specific tools [88, 90] or Unified Modeling Language (UML) [99, 100, 101]. UML is also used to create Domain Specific Languages (DSL) [86, 87, 89, 91, 92]. DSLs are created with a domain specific focus, wireless sensor network in this case. Our MDSE approach uses a formal verification tool (CPN tool) as the modeling tool for the protocols. This allows for an additional step of verification and simulation obtaining platform specific models. One other similar approach [102] performs code generation using CPN tools, which creates a dedicated tool for code generation. In our approach, we use the existing PetriCode [49] tool for model transformations and code generation for platform specific models for MiXiM and TinyOS. PetriCode allows for code generation for multiple platforms using platform specific plugins as templates. Additionally, we can also exploit common features of multiple simulation platforms like MiXiM and Castalia [69] or multiple hardware platforms like TinyOS and ContikiOS [82]. This allows us to extend the MDSE approach for multiple platforms with minimal effort.

The code generation approach is semi-automatic. The executable solution at the platform end needs to be suitably modified to suit user configuration requirements, only default configurations can be provided. The code generation templates created for one MAC protocol can be re-used to create other MAC protocols, but would need modification to make up the difference between the protocols. The handling of a particular operation, say sleep, would differ in different MAC protocols. The creator of specific code generation templates needs a sufficient understanding of the platform and a generic understanding of the CPN models.

Chapter 6

Conclusion and Future Work

In this chapter, we summarize the contributions of the thesis, provide the main conclusions, and outline directions for future work. In addition, we discuss how the results of the thesis can be generalized beyond the domain of Industrial Wireless Sensor Networks (IWSN).

6.1 Summary of Contributions

We proposed a Dual-Mode Adaptive Medium Access Control (DMAMAC) protocol for process control applications in IWSN. IWSN solutions are used to automate the process control operations. The novelty of the DMAMAC protocol is to exploit the process control feature of steady and transient states, and integrating it into the protocol operation as dual-mode of operation (transient and steady mode). The transient mode supports the data intensive transient state, and the steady mode exploits the relatively less data intense steady state to provide an energy efficient solution. In particular, this makes it possible to provide a feasible energy efficient solution to the process control application, while at the same time satisfying stringent real-time requirements imposed by the application domain. The typical real-time requirements are low-delay, reliable and predictable communication. We assume process control scenarios with a dominant steady state. We thoroughly analyzed the proposed DMAMAC protocol to obtain a validated, verified, and performance analyzed design. Protocol evaluation was performed in separate steps as described in the individual chapters in the thesis and briefly summarised below.

Initial analytical evaluation In the initial article [50] proposing the DMAMAC protocol, we presented a first analytical evaluation, in order to demonstrate that the DMAMAC protocol is potentially energy efficient. This constituted an initial evaluation of the protocol design before proceeding to further evaluation. The DMAMAC protocol gains in energy efficiency compared to the GinMAC protocol was based on the length of the steady superframe. The longer the steady superframe is, the higher are the energy savings.

Simulation Simulation is a general approach for performance analysis of protocol designs in the network domain. A simulation model of the DMAMAC protocol was created in MiXiM, a tool based on the OMNeT++ framework. The model created was used in a network simulation to evaluate the performance of the DMAMAC protocol, with a configuration of 25 nodes. Two variants of the DMAMAC protocol, hybrid and TDMA were obtained as a result of the lessons learned from the performance evaluation study. The two DMAMAC variants have distinct advantages and disadvantages, and are hence applicable in different scenarios.

Verification Formal verification is a process of ensuring the proper functioning of the protocol, and that the protocol adheres to its requirements. We verified the DMAMAC protocol design and functioning using timed automata models created using the Uppaal tool. Also, real-time properties were verified using several variants of models of the DMAMAC protocol. This increased our confidence in the design of the DMAMAC protocol. Also, we used the verification model to validate the models used for simulation and deployment.

Implementation and Deployment The DMAMAC protocol was implemented using the TinyOS operating system. The DMAMAC protocol along with other off the shelf components for time-synchronization, radio module, and an application module was implemented using the nesC programming language. The implementation was deployed on the Zolertia Z1 hardware platform to perform experiments in two separate scenarios. The scenarios included testing of the network operating on the

DMAMAC protocol in a motor room with running motors. The key point was to study interference similar to what can be found in typical process control scenarios. The experiments resulted in positive results for the DMAMAC protocol operation in a scenario similar to industrial process control. In particular, the notification packet failure was low leading to low-switch failure. The DMAMAC protocol includes reliability features such as, re-transmission, repetition of notification packets, and robust superframe design. The robust superframe design allows nodes that are in the wrong mode with respect to the rest of the network, to synchronize back to the correct mode.

It follows from the above that we have performed a comprehensive design and evaluation of the developed DMAMAC protocol.

Lastly, we re-visited the development approach used for the DMAMAC protocol. The DMAMAC protocol artifacts (simulation model, verification model, and implementation model) created for multiple platforms were largely disconnected. The protocol specification is used to create a platform specific model, which is then used to create the final code. We proposed a model-based development approach with the aim of creating closely linked software artifacts, and reducing the development time by automating the code generation process for the different platforms. The model-based development approach uses a formal verification tool, CPN tools as a starting platform. A MAC protocol model was created in CPN tools, and verified. Initial simulations are performed, resulting in an improved protocol design. The refined CPN model is then used as an input to the PetriCode tool, which is used to generate platform specific implementation code. In our implemented approach, we generate implementation code for the MiXiM platform and the TinyOS platform. We created code generation templates which are used as plugins to the PetriCode tool. The generated code is used via simulation for the MiXiM code and via deployment for the TinyOS code. We used the GinMAC protocol as an example to present the working of the model-based approach, and successfully generated implementation code for both target platforms and analyzed the protocol performance.

6.2 Re-visiting Research Questions

- Can MAC protocols for wireless process control be made energy efficient by exploiting intrinsic properties of the process control domain?
- What guarantees can be provided with respect to real-time operation and reliability?
- How can model-driven engineering techniques be used for MAC protocol development?

We created the energy efficient DMAMAC protocol for process control applications by exploiting the property of alternating between the states, steady and transient. The DMAMAC protocol consists of the dual-modes to serve the process control states. We simulated, verified, and implemented the DMAMAC protocol on a real hardware platform to assess the protocol performance. We evaluated energy efficiency, network lifetime, reliability (via notification packet failure), while also keeping the real-time operation at par with the related GinMAC protocol. The simulation evaluation shows a switch failure of ≤ 0.23 for target configurations, which is a tolerable value. We proposed a model-driven development approach for the development of MAC protocols. This was also implemented with platform specific code generation for multiple platforms, simulation in MiXiM, and implementation in TinyOS. The research questions have been successfully addressed in the thesis.

6.3 Beyond IWSN Process Control

The DMAMAC protocol is an application specific protocol designed for process control applications. The design of the dual-mode in the DMAMAC protocol is inherently based on the steady and transient operation of a process control application. This idea could be generalized and applied in other domains such as health care and home/building automation. The health care domain has stringent application requirements similar to the IWSN domain [103]. Wireless Body Area Networks [104, 105, 106] (WBANs) are applicable in scenarios similar to IWSN: monitoring, and monitoring with control [104].

An example is glucose monitoring and control [107]. Here, a sensor tracks the glucose levels of a patient and an insulin actuator injects insulin whenever required. This scenario is similar to the temperature tracking and control example of the process control scenario. Emergency alert systems are related to health care monitoring. The DMAMAC protocol has its own alert message mechanism, which is applicable also in a health care monitoring setting. Given that the DMAMAC protocol, apart from being energy efficient also satisfies stringent real-time requirements, it is a relevant candidate for a protocol in the health care domain.

The home and building automation consists of temperature monitoring and control, air monitoring and control. These are also similar scenarios, where a physical quantity is tracked and controlled. Thus, the features of the DMAMAC protocol can be applied to also home and building automation.

The model-based verification model created in Uppaal [52] for the qualitative verification of the DMAMAC protocol is generic in its design. The model design builds on a generic representation of Medium Access Control (MAC) protocols with locations (states) common to most MAC protocols (sleep, send data, receive data). The additional operations of the DMAMAC protocol are modelled as separate locations (states), for the alert message operation. The basic model can be extended to provide other MAC operations, including Carrier Sense Multiple Access operations such as backoff, request to send, and clear to send. Thus, in general, the Uppaal model created for the DMAMAC protocol is generic, re-usable, and extendable.

The model-based development approach proposed in this thesis, is an alternative development approach for MAC protocols in IWSN. The development approach is currently targeted at MAC protocols, but can be extended to protocols at other layers including the application and the network layer. The programming structure of the application protocols, and the network protocols, in the target platforms MiXiM and TinyOS are largely similar to the MAC protocols. This similarity can be exploited to re-use code generation templates created for the MAC protocol code generation. This makes the underlying idea and approach applicable to a wider set of communication protocols.

6.4 Future Work

One direction of future work is related to the protocol design for Industrial Wireless Sensor Networks (IWSN). The second direction is related to model-based development approaches for protocols. Also, during the work on this thesis, we have successfully realized the future work proposed in [50, 51, 52, 55].

The DMAMAC protocol contains features that are applicable towards other domains, as discussed in the previous section. It is therefore an interesting direction to study the performance and deployment of the DMAMAC protocol in other domains, and making suitable adjustments to the design to make it applicable to either the health care and the home/building automation domain.

One other interesting direction of future work is to analyze the performance of existing wireless standards with the DMAMAC protocol substituting the current MAC protocol (in the wireless standard). The existing wireless standards, wirelessHART [9] and ISA100.11a [10] are more of a framework than a complete solution. The wireless standards are particularly proposed for the process automation domain. These wireless standards contain Medium Access Control (MAC) protocols embedded in the framework. The MAC protocol can either be replaced by the DMAMAC protocol, or the features of the DMAMAC protocol can be embedded into the existing MAC. These alternative solutions can be compared with existing MAC solutions proposed for these frameworks.

The design and performance of the DMAMAC protocol were analyzed for energy efficiency, network life time, and switch failure. The protocol can be further analyzed for requirements including scalability and security. Industrial installations are likely to consist of networks larger than the ones used for the DMAMAC protocol testing. Investigating the scalability limits of the DMAMAC protocol could provide useful insights. Industrial network is susceptible to security attacks both passive and active. Different wireless standards like wirelessHART and ISA100.11a have dedicated devices to handle security. The DMAMAC protocol could benefit from such external dedicated security infrastructure, or alternatively, security can be embedded in the MAC protocol operation. A direction of future work can include embedding security measures within the DMAMAC protocol operation.

We evaluated the deployment of a network operating on the DMAMAC protocol. The deployment test was conducted in a motor lab with a process control like scenario. One important direction is deployment testing in a real process control scenario in industry. This would provide us information on the DMAMAC protocol performance on wireless channel conditions in a process control industry. Also, the effect of switch failures on the operation of the process would be interesting to investigate further in such a setting.

The verification model created in the Uppaal tool is currently non-stochastic. The Uppaal tool allows for stochastic verification of models. This can be used for evaluating MAC protocols like the DMAMAC protocol for stochastic properties. Uncertainties like packet error, number of switches that happen in a given duration, and switch failures can be assessed using a stochastic model of the DMAMAC protocol. Also, the effect of these parameters on the energy efficiency of the protocol can be investigated.

The model-based development approach proposed in this thesis is designed using the GinMAC protocol as a running example. The DMAMAC protocol design is built upon the GinMAC protocol features. A direction of future work is to test the model-based development approach applied directly on the DMAMAC protocol. This would also allow us to compare the generated code for the MiXiM-OMNeT++ platform with the simulation model created manually based on requirements. Similarly, comparison of the TinyOS code generation with the manually created version of the DMAMAC protocol would provide insights to the advantages of using the model based approach. Further, the PetriCode tool takes a CPN model and a plugin code template as an input to generate platform specific code. Additional code templates can be added to generate implementation code for platforms like the Castalia simulator platform and the ContikiOS deployment platform.

Bibliography

[1] I. Khemapech, I. Duncan, and A. Miller. A survey of Wireless Sensor Networks technology. In *Proceedings of the 6th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*, 2005.

[2] K. Romer and F. Mattern. The design space of Wireless Sensor Networks. *IEEE Wireless Communications*, 11(6):54–61, 2004. ISSN 1536-1284.

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A survey. *Computer Networks*, 38(4):393–422, 2002. ISSN 1389-1286.

[4] Xingfa Shen, Zhi Wang, and Youxian Sun. Wireless Sensor Networks for Industrial Applications. In *Proceedings of the 5th World Congress on Intelligent Control and Automation*, volume 4, pages 3636–3640, 2004.

[5] V Çağrı Güngör and Gerhard P Hancke. *Industrial Wireless Sensor Networks: Applications, Protocols, and Standards*. CRC Press, 2013.

[6] V.C. Gungor and G.P. Hancke. Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches. *IEEE Transactions on Industrial Electronics*, 56(10):4258–4265, 2009. ISSN 0278-0046.

[7] A. Ajith Kumar S., K. Øvsthus, and L.M. Kristensen. An Industrial Perspective on Wireless Sensor Networks: A survey of Requirements, Protocols, and Challenges. *IEEE Communications Surveys Tutorials*, 16(3):1391–1412, 2014. ISSN 1553-877X.

[8] Xiaomin Li, Di Li, Jiafu Wan, Athanasios V. Vasilakos, Chin-Feng Lai, and Shiyong Wang. A review of Industrial Wireless Networks in the context of Industry 4.0. *Wireless Networks*, pages 1–19, 2015. ISSN 1572-8196.

[9] Deji Chen, Mark Nixon, and Aloysius Mok. *WirelessHART: Real-Time Mesh Network for Industrial Automation*. Springer, 1st edition, 2010.

[10] ISA100 Wireless Working Group. Draft standard ISA100. 11a. In *Internal working draft*. International Society of Automation, May 2008.

[11] Zigbee Alliance. Zigbee specification. In *Zigbee Document 053474r13*. Zigbee Alliance, May 2008.

[12] *Industrial Communication Networks –Fieldbus Specifications –WIA –PA Communication Network and Communication Profile*, 2009.

[13] V.C. Gungor, Bin Lu, and G.P. Hancke. Opportunities and Challenges of Wireless Sensor Networks in Smart Grid. *IEEE Transactions on Industrial Electronics*, 57(10):3557–3564, 2010. ISSN 0278-0046.

[14] M.R. Akhondi, A. Talevski, S. Carlsen, and S. Petersen. Applications of Wireless Sensor Networks in the Oil, Gas and Resources Industries. In *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 941–948, April 2010.

[15] Kewei Sha, Weisong Shi, and O. Watkins. Using Wireless Sensor Networks for Fire Rescue Applications: Requirements and Challenges. In *IEEE International Conference on Electro/information Technology*, pages 239–244, 2006.

[16] Andrey Somov, Alexander Baranov, Denis Spirjakin, Andrey Spirjakin, Vladimir Sleptsov, and Roberto Passerone. Deployment and Evaluation of a Wireless Sensor Network for methane leak detection. *Sensors and Actuators A: Physical*, 202:217–225, 2013. ISSN 0924-4247.

[17] Andrey Somov, Alexander Baranov, and Denis Spirjakin. A Wireless Sensor Actuator system for hazardous gases detection and control. *Sensors and Actuators A: Physical*, 210:157–164, 2014. ISSN 0924-4247.

[18] A. Flammini, D. Marioli, E. Sisinni, A. Taroni, and M. Pezzotti. A Wireless Thermocouples Network for temperature control in plastic machinery. In *IEEE International Workshop on Factory Communication Systems*, pages 219–222, 2006.

[19] Ramazan Bayindir and Yucel Cetinceviz. A water pumping control system with a Programmable Logic Controller (PLC) and Industrial Wireless Modules for Industrial Plants: An Experimental Setup. *ISA Transactions*, 50(2):321–328, 2011. ISSN 0019-0578.

[20] Brendan Galloway and Gerhard P Hancke. Introduction to Industrial Control Networks. *IEEE Communications Surveys & Tutorials*, 15(2):860–880, 2013. ISSN 1553-877X.

[21] A. Willig, K. Matheus, and A. Wolisz. Wireless technology in industrial networks. *Proceedings of the IEEE*, 93(6):1130–1151, June 2005. ISSN 0018-9219.

[22] G. Anastasi, M. Conti, and M. Di Francesco. A Comprehensive Analysis of the MAC Unreliability Problem in IEEE 802.15.4 Wireless Sensor Networks. *IEEE Transactions on Industrial Informatics*, 7 (1):52–65, 2011. ISSN 1551-3203.

[23] Andreas F Molisch, Kannan Balakrishnan, Dajana Cassioli, Chia-Chin Chong, Shahriar Emami, Andrew Fort, Johan Karedal, Juergen Kunisch, Hans Schantz, Ulrich Schuster, et al. IEEE 802.15. 4a channel model-final report. *IEEE P802*, 15(04), 2004.

[24] CC2420 Datasheet. 2.4 ghz ieee 802.15. 4 zigbee-ready rf transceiver. *Chipcon products from Texas Instruments*, 2006.

[25] IETF. IEEE 802.15.4e. https://standards.ieee.org/findstds/standard/802.15.4e-2012.html, 2012.

[26] Domenico De Guglielmo, Giuseppe Anastasi, and Alessio Seghetti. From IEEE 802.15.4 to IEEE 802.15. 4e: A step towards the Internet of Things. In *Advances onto the Internet of Things*, pages 135–152. Springer, 2014.

[27] Feng Chen, R. German, and F. Dressler. Towards IEEE 802.15.4e: A study of performance aspects. In *Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 68–73, 2010.

[28] K Pister and Lance Doherty. Tsmp: Time Synchronized Mesh Protocol. In *Proceedings of the IASTED International Symposium Distributed Sensor Networks*, pages 391–398, 2008.

[29] I. Howitt and J.A. Gutierrez. IEEE 802.15.4 low rate - Wireless Personal Area Network coexistence issues. In *IEEE Wireless Communications and Networking*, volume 3, pages 1481–1486, 2003.

[30] L. Lo Bello and E. Toscano. Coexistence Issues of Multiple Co-located IEEE 802.15.4/zigbee Networks Running on Adjacent Radio Channels in Industrial Environments. *IEEE Transactions on Industrial Informatics*, 5(2):157–167, 2009. ISSN 1551-3203.

[31] The international society of automation, 2016.

[32] Li Da Xu, Wu He, and Shancang Li. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014. ISSN 1551-3203.

[33] S. Petersen and S. Carlsen. WirelessHART Versus ISA100.11a: The Format War Hits the Factory Floor. *IEEE Industrial Electronics Magazine*, 5(4):23–34, 2011. ISSN 1932-4529.

[34] Felix Büsching, Wolf-Bastian Pöttner, Dieter Brökelmann, Georg von Zengen, Robert Hartung, Karsten Hinz, and Lars Wolf. A demonstrator of the GINSENG-approach to performance and closed loop control in WSNs. In *Networked Sensing Systems (INSS), 2012 Ninth International Conference on*, pages 1–2, 2012.

[35] Nurcan Tezcan and Wenye Wang. ART; An Asymmetric and Reliable Transport Mechanism for Wireless Sensor Networks. *International Journal of Sensor Networks*, 2(3/4):188–200, 2007. ISSN 1748-1279.

[36] Vehbi Cagri Gungor, Özgür B. Akan, and Ian F. Akyildiz. A Real-Time and Reliable Transport $(RT)^2$ Protocol for Wireless Sensor and Actor Networks. *IEEE/ACM Transactions in Networks*, 16(2):359–370, 2008. ISSN 1063-6692.

[37] Hongwei Zhang, Anish Arora, Young-Ri Choi, and Mohamed G. Gouda. Reliable Bursty Convergecast in Wireless Sensor Networks. *Computer Communications*, 30(13):2560–2576, 2007. ISSN 0140-3664.

[38] E. Felemban, Chang-Gun Lee, and E. Ekici. MMSPEED: Multipath Multi-SPEED protocol for QoS guarantee of reliability and timeliness in Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, 5(6):738–754, 2006. ISSN 1536-1233.

[39] A. Manjeshwar and D.P. Agrawal. TEEN: A routing protocol for enhanced efficiency in Wireless Sensor Networks. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, IPDPS, 2001.

[40] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Stuik, JP. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. *Internet Draft*, 2012. ISSN 2080-1721.

[41] A. Bachir, M. Dohler, T. Watteyne, and K. K. Leung. MAC Essentials for Wireless Sensor Networks. *IEEE Communications Surveys Tutorials*, 12(2):222–248, 2010. ISSN 1553-877X.

[42] V Sachan, S Imam, and M Beg. Energy-efficient communication methods in Wireless Sensor Networks: A critical review. *International Journal of Computer Applications*, 39(17), 2012.

[43] Injong Rhee, Ajit Warrier, Mahesh Aia, Jeongki Min, and Mihail L. Sichitiu. Z-MAC: A Hybrid MAC for Wireless Sensor Networks. *IEEE/ACM Transactions on Networks*, 16(3):511–524, 2008. ISSN 1063-6692.

[44] Petcharat Suriyachai, James Brown, and Utz Roedig. Time-Critical Data Delivery in Wireless Sensor Networks. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems*, volume 6131, pages 216–229. Springer, 2010. ISBN 978-3-642-13650-4.

[45] P. Suriyachai, U. Roedig, and A. Scott. Implementation of a MAC protocol for QoS support in Wireless Sensor Networks. In *Proceedings of the IEEE Conference on Pervasive Computing and Communications*, pages 1–6, march 2009.

[46] S.C. Ergen and P. Varaiya. PEDAMACS: Power Efficient and Delay Aware Medium Access Protocol for Sensor Networks. *Mobile Computing, IEEE Transactions on*, 5(7):920–930, 2006. ISSN 1536-1233.

[47] L. Sitanayah, C.J. Sreenan, and K.N. Brown. ER-MAC: A Hybrid MAC Protocol for Emergency Response Wireless Sensor Networks. In *Proceedings of the Fourth International Conference on Sensor Technologies and Applications (SENSORCOMM)*, pages 244–249, 2010.

[48] Adrian Perrig, John Stankovic, and David Wagner. Security in Wireless Sensor Networks. *ACM Communications*, 47(6):53–57, 2004. ISSN 0001-0782.

[49] Kent Inge Fagerland Simonsen. Petricode: a tool for template-based code generation from CPN models. In *Software Engineering and Formal Methods*, pages 151–163. Springer, 2013.

[50] A. Ajith Kumar S., Knut Øvsthus, and Lars M. Kristensen. Towards a Dual-Mode Adaptive MAC Protocol (DMA-MAC) for Feedback-based Networked Control Systems. *Procedia Computer Science*, 34: 505–510, 2014. ISSN 1877-0509. The 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops.

[51] A. Ajith Kumar S., Lars M. Kristensen, and Knut Øvsthus. Simulation-based Evaluation of DMA-MAC: A Dual-mode adaptive MAC Protocol for Process Control. In *Proceedings of the 8th International Conference on Simulation Tools and Techniques*, SIMUTools '15, pages 218–227. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015. ISBN 978-1-63190-079-2.

[52] A. Ajith Kumar S., Andreas Prinz, and Lars M Kristensen. Model-based Specification and Validation of the DMAMAC Protocol. *International Journal of Critical Computer Based System*, x:0–0, 2017. ISSN (Submitted).

[53] Kim G Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[54] A. Ajith Kumar S., Knut Øvsthus, and Lars M. Kristensen. Implementation and Deployment Evaluation of the DMAMAC Protocol for Wireless Sensor Actuator Networks. In *Proceedings of the The 7th International Conference on Ambient Systems, Networks and Technologies*, volume 83 of *ANT 2016*, pages 329–336, 2016.

[55] A. Ajith Kumar S. and Kent I. F. Simonsen. Towards a Model-based Development Approach for Wireless Sensor-Actuator Network Protocols. In *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, CyPhy, pages 35–39, 2014. ISBN 978-1-4503-2871-5.

[56] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2013.

[57] A. Ajith Kumar S. and Kent I. F. Simonsen. Model-based Development for MAC Protocols in Industrial Wireless Sensor Networks. In *Proceedings of the International Workshop on Petri Nets and Software Engineering*, PNSE. CEUR Workshop Proceedings, 2016.

[58] A. Ajith Kumar S. Artifacts for simulation code and implementation code. `http://home.hib.no/ansatte/aaks/`, 2016. Last Accessed: 31-05-2016.

[59] T. O Donovan, J. Brown, U. Roedig, C.J. Sreenan, J. do O, A. Dunkels, A. Klein, J.S. Silva, V. Vassiliou, and L. Wolf. GINSENG: Performance Control in Wireless Sensor Networks. In *Proceedings of the 7th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks*, pages 1–3, 2010.

[60] Pangun Park, C. Fischione, A. Bonivento, K.H. Johansson, and A. Sangiovanni-Vincent. Breath: An Adaptive Protocol for Industrial Control Applications using Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, 10(6):821–838, 2011. ISSN 1536-1233.

[61] Wei Shen, Tingting Zhang, F. Barac, and M. Gidlund. PriorityMAC: A Priority-Enhanced MAC Protocol for Critical Traffic in Industrial Wireless Sensor and Actuator Networks. *IEEE Transactions on Industrial Informatics*, 10(1):824–835, 2014. ISSN 1551-3203.

[62] E. Weingartner, H. vom Lehn, and K. Wehrle. A performance comparison of recent Network Simulators. In *IEEE International Conference on Communications*, pages 1–5, 2009.

[63] Harsh Sundani, Haoyue Li, Vijay Devabhaktuni, Mansoor Alam, and Prabir Bhattacharya. Wireless Sensor Network simulators a survey and comparisons. *International Journal of Computer Networks*, 2 (5):249–265, 2011.

[64] Andras Varga. OMNeT++. In *Modeling and Tools for Network Simulation*, pages 35–59. Springer, 2010.

[65] Xiaodong Xian, Weiren Shi, and He Huang. Comparison of OMNeT++ and other simulator for WSN simulation. In *3rd IEEE Conference on Industrial Electronics and Applications*, pages 1439–1443, 2008.

[66] Z. Zhang, Z. Lu, Q. Chen, X. Yan, and L. R. Zheng. COSMO: Co-simulation with MATLAB and OMNeT++ for Indoor Wireless Networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–6, 2010.

[67] Teerawat Issariyakul and Ekram Hossain. *Introduction to network simulator NS2*. Springer, 2011.

[68] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Klein Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating Wireless and Mobile Networks in OMNeT++ the MiXiM vision. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 71:1–71:8, 2008.

[69] Athanassios Boulis. Castalia user manual. *Online: http://castalia. npc. nicta. com. au/pdfs/Castalia-User Manual. pdf*, 2009.

[70] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic Symbolic Model Checker. In *Computer performance evaluation: modelling techniques and tools*, pages 200–204. Springer, 2002.

[71] Ansgar Fehnker, Rob Van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann, and Wee Lum Tan. Automated analysis of AODV using Uppaal. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 2012. ISBN 978-3-642-28755-8.

[72] Simon Tschirner, Liang Xuedong, and Wang Yi. Model-based Validation of QoS properties of Biomedical Sensor Networks. In *Proceedings of the 8th ACM International Conference on Embedded Software*, pages 69–78. ACM, 2008. ISBN 978-1-60558-468-3.

[73] Ansgar Fehnker, Lodewijk van Hoesel, and Angelika Mader. Modelling and Verification of the LMAC protocol for Wireless Ssensor Networks. In *Integrated Formal Methods*, volume 4591 of *Lecture Notes in Computer Science*, pages 253–272. Springer, 2007. ISBN 978-3-540-73209-9.

[74] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2): 183–235, 1994.

[75] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on Uppaal. In *Formal methods for the design of real-time systems*, pages 200–236. Springer, 2004.

[76] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[77] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In Werner Weber, JanM. Rabaey, and Emile Aarts, editors, *Ambient Intelligence*, pages 115–148. Springer, 2005.

[78] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. *SIGPLAN*, 38(5):1–11, May 2003. ISSN 0362-1340.

[79] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 39–49, 2004.

[80] Z1 datasheet, zolertia. `http://www.zolertia.io//`, 2016. Accessed: 25-01-2016.

[81] JeongGil Ko, Nicolas Tsiftes, Adam Dunkels, and Andreas Terzis. Pragmatic low-power interoperability: Contikimac vs tinyos lpl. In *Sensor, mesh and ad hoc communications and networks (SECON), 2012 9th annual IEEE communications society conference on*, pages 94–96. IEEE, 2012.

[82] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462. IEEE, 2004.

[83] CC2560 Datasheet. Simplelink multi-standard 2.4 ghz ultra-low power wireless mcu. *Chipcon products from Texas Instruments*, 2006.

[84] Richard Soley et al. Model Driven Architecture. *OMG white paper*, 308(308):5, 2000.

[85] Anneke G Kleppe, Jos B Warmer, and Wim Bast. *MDA explained: the Model Driven Architecture: practice and promise*. Addison-Wesley Professional, 2003.

[86] Fernando Losilla, Cristina V Chicote, Barbara Alvarez, Andres Iborra, and Pedro Sanchez. Wireless Sensor Network Application Development: An Architecture-Centric MDE approach. In *Software Architecture*, volume 4758 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2007. ISBN 978-3-540-75131-1.

[87] Cristina Vicente-Chicote, Fernando Losilla, Barbara Alvarez, Andres Iborra, and Pedro Sanchez. Applying MDE to the development of flexible and reusable Wireless Sensor Networks. *International Journal of Cooperative Information Systems*, 16(3-4):393–412, 2007.

[88] M. M R Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri. A Framework for Modeling, Simulation and Automatic Code Generation of Sensor Network Application. In *Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 515–522, 2008.

[89] T. Rodrigues, T. Batista, F.C. Delicato, P.F. Pires, and A.Y. Zomaya. Model-driven approach for building efficient Wireless Sensor and Actuator Network Applications. In *Proceedings of the International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 43–48, 2013.

[90] Pruet Boonma and Junichi Suzuki. Model-driven Performance Engineering for Wireless Sensor Networks with Feature Modeling and Event Calculus. In *Proceedings of the 3rd Workshop on Biologically inspired Algorithms for Distributed Systems (BADS)*, pages 17–24. ACM, 2011.

[91] K. Doddapaneni, E. Ever, O. Gemikonakli, I. Malavolta, L. Mostarda, and H. Muccini. A model-driven engineering framework for architecting and analysing wireless sensor networks. In *Proceedings of the International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 1–7, 2012.

[92] Ryo Shimizu, Kenji Tei, Yoshiaki Fukazawa, and Shinichi Honiden. Model driven development for rapid prototyping and optimization of wireless sensor network applications. In *Proceedings of the International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 31–36, 2011.

[93] Wei Ye, J. Heidemann, and D. Estrin. Medium Access Control with coordinated adaptive sleeping for Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, 2004. ISSN 1063-6692.

[94] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 95–107. ACM, 2004. ISBN 1-58113-879-2.

[95] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, pages 307–320. ACM, 2006. ISBN 1-59593-343-3.

[96] Tijs van Dam and Koen Langendoen. An Adaptive Energy-efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 171–180. ACM, 2003. ISBN 1-58113-707-9.

[97] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.

[98] Nguyen Xuan Thang, Michael Zapf, and Kurt Geihs. Model driven development for data-centric sensor network applications. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, pages 194–197. ACM, 2011.

[99] Gerhard Fuchs and Reinhard German. UML2 Activity Diagram based Programming of Wireless Sensor Networks. In *Proceedings of the International Workshop on Software Engineering for Sensor Network pplications*, pages 8–13. ACM, 2010. ISBN 978-1-60558-969-5.

[100] M. Mura and M. G. Sami. Code Generation from Statecharts: Simulation of Wireless Sensor Networks. In *Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, pages 525–532, 2008.

[101] Hiroshi Wada, Pruet Boonma, Junichi Suzuki, and Katsuya Oba. Modeling and executing Adaptive Sensor Network applications with the Matilda UML virtual machine. In *Proceedings of the 11th IASTED International Conference on Software Engineering and Applications*, pages 216–225. ACTA Press, 2007.

[102] Vegard Veiset and Lars Michael Kristensen. Transforming platform independent cpn models into code for the tinyos platform: A case study of the rpl protocol. In *PNSE+ ModPE*, pages 259–260. Citeseer, 2013.

[103] Maulin Patel and Jianfeng Wang. Applications, Challenges, and Prospective in Emerging Body Area Networking Technologies. *IEEE Wireless Communications Magazine*, 17(1):80–88, 2010.

[104] Hande Alemdar and Cem Ersoy. Wireless Sensor Networks for healthcare: A survey. *Computer Networks*, 54(15):2688–2710, 2010. ISSN 1389-1286.

[105] Benoît Latré, Bart Braem, Ingrid Moerman, Chris Blondia, and Piet Demeester. A survey on wireless body area networks. *Wireless Networks*, 17(1):1–18, January 2011. ISSN 1022-0038.

[106] Min Chen, Sergio Gonzalez, Athanasios Vasilakos, Huasong Cao, and Victor C. M. Leung. Body Area Networks: A Survey. *Mobile Networks and Applications*, 16(2):171–193, 2011. ISSN 1572-8153.

[107] D. M. Barakah and M. Ammad-uddin. A Survey of Challenges and Applications of Wireless Body Area network (WBAN) and Role of a Virtual Doctor Server in Existing Architecture. In *Third International Conference on Intelligent Systems Modelling and Simulation*, pages 214–219, 2012.

Part II

Articles

Chapter 7

Towards a Dual-mode Adaptive MAC Protocol (DMA-MAC) for Feedback-based Networked Control Systems

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2014) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

The 2nd International Workshop on Communications and Sensor Networks (ComSense-2014)

Towards a Dual-Mode Adaptive MAC Protocol (DMA-MAC) for Feedback-based Networked Control Systems

Admar Ajith Kumar Somappa*[a,b], Knut Øvsthus[a], Lars Michael Kristensen[a]

[a] *Faculty of Engineering and Business Administration, Bergen University College, Norway*
[b] *Faculty of Engineering and Science, University of Agder, Norway*

Abstract

Automated control systems play an important part in many industrial domains and the medium used for communication between devices in these systems is in transition from wired to wireless for cost reasons. Control systems have strict requirements on delay, throughput, and reliability, that vary with time during operation. Addressing these requirements requires predictable and robust protocols to be employed, and they must be adaptive to the varying states of the controlled process. In this article, we propose a dual-mode adaptive medium access control protocol that caters for two main operation modes in control systems: the steady mode operation, and the transient mode operation. We present initial performance analysis results focusing on energy consumption, including a comparison with a related single-mode industrial monitoring and control protocol.

Keywords: Wireless Sensor Actuator Networks, Process Control, Automation, Industrial Application.

1. Introduction

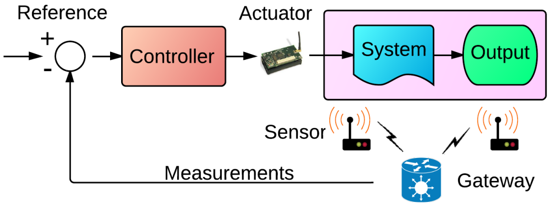


Fig. 1: Feedback based control loop

A Wireless Sensor Actuator Network (WSAN)[1] consists of sensors and actuators that use radios to send, relay, and receive information. WSANs are applied across several domains including process and factory automation. The use of WSANs in these application domains has had a significant effect on the process industry, by reducing operating costs and increasing automation. With actuators being part of the system, feedback-based control-loop automation is one of the main applications. Feedback-based control systems that use wired or wireless solutions for data transfer are known as Networked Control Systems (NCSs)[2]. A general control system consists of a reference input, plant output, control input, sensors, actuators, and a controller and a typical feedback-based control loop is shown in figure 1. Traditionally, wired communication has been used in NCSs to achieve high reliability, low delay, and high bandwidth. With the advancement in WSAN technology, industry has started utilizing wireless solutions for NCSs. There are several challenges that are to be addressed when implementing wireless solutions for process control applications[3]. The primary issues that arise as a result of switching from a wired to wireless medium are packet loss,

* Corresponding author. Tel.: +47 55 58 75 88
E-mail address: aaks@hib.no

reduced bandwidth, and increased latency. Considering the fact that nodes in WSAN are generally battery powered, energy efficiency is also an important factor in addition to satisfying real-time requirements.

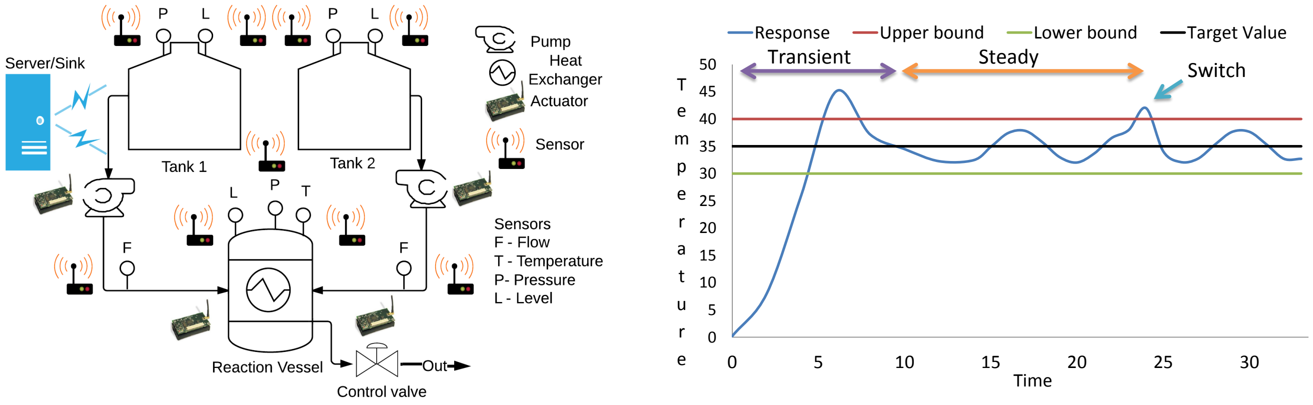
In this article, we focus on process control systems based on wireless communication. Process control generally has two states: steady and transient. In the transient state, data values vary rapidly with time, generating large amounts of traffic. Once the data values start stabilizing, the process is said to be in steady state, and the amount of data traffic required is then low. Thus, we propose a medium access control (MAC) protocol designed to have two operational modes to cater for varying process control states. The protocol is called the Dual-Mode Adaptive MAC protocol (DMA-MAC) which defines operation modes based on the two process states. A transient mode that supports the transient state and a steady mode to support steady state. The sensors communicate less frequently during the steady mode thus saving energy and reducing interference to other operations in the vicinity. In a wireless sensor node, the transceiver consumes far more energy than the micro-controller[4]. Thus, by reducing the communication of information during steady state, energy consumption can be reduced. The network architecture considered, consists of a sink node, sensor nodes, and actuator nodes in a tree topology. With sensor/actuator nodes at the lowest level known as leaf nodes, and the parent/child nodes at organized higher levels. The sink node is assumed to be wire powered and is powerful relative to the sensors and actuators. In DMA-MAC operation, the switch between the operational modes is important. In case the measurement crosses a certain threshold (steady state error interval), sensors will notify the sink of this change, and the sink will take the necessary steps to initiate transient mode operation. The switch from transient to steady mode is decided by the sink, and the sensors are notified. The switch decision is made based on the underlying process model, which captures the characteristic features of the process. The main advantage of having a dual-mode adaptive protocol over single-mode protocols is for the case with process models that spends less time in transient state than in the steady state. For such a case the DMA-MAC protocol consumes far less energy than a single-mode protocol. Key design considerations and assumptions used for the design of DMA-MAC are as follows: The setting of thresholds for the sensors to detect the switch is assumed to be based on the process model. For multi-variable input models, we design the transient mode operation to continue until the slowest of the input reaches steady state.

1.1. Important features of DMA-MAC

The aim of our protocol in comparison to existing protocols is to preserve the key performance metrics such as reliability, throughput, low delay, and at the same time improve energy efficiency. The main features that the proposed protocol focuses on, and the overhead caused due to the design are as follows: *Predictability*: process control system procedures have predictable performance since they are pre-calculated to ensure smooth operation. Simulations are performed based on the given process model, to ensure proper design, and to test for predictable performance. From the communication part, with the usage of TDMA based superframes, DMA-MAC has highly predictable performance for the wireless communication. *Reliability*: with the use of reliability mechanisms similar to GinMAC[5] (a single-mode protocol), reliability is maintained. *Energy conservation*: in the steady mode DMA-MAC protocol has lower data transmission. We propose this protocol for process control scenario where the steady state is dominant compared to the transient state, and hence we have lower energy consumption compared to protocols having single-mode of operation. This is due to the fact that protocols with single-mode of operation should be adhering to transient mode operation at all times to have the required response in transient state. *Overhead*: DMA-MAC protocl has a complex protocol structure due to the dual-mode operation, which results in larger code space, and hence affects hardware performance. Also, the need to ensure efficient state switch detection, and relay within a given duration, is especially critical for state switch from steady to transient.

2. Related Work

GinMAC[5] is a protocol proposed in the GINSENG[6] project for process monitoring systems. The main features of this protocol are *Offline dimensioning*, *Exclusive TDMA* (Time Division Multiple Access) and *Delay Conform Reliability Control*. The design of GinMAC was aimed at satisfying real-time constraints for sensors and actuators. The TDMA structure and offline dimensioning makes the protocol predictable. We use GinMAC protocol as a basis for our protocol design and then also for comparison. Breath[7] is a protocol aimed at reducing energy consumption and providing delay guarantees. Breath is based on Carrier Sense Multiple Access (CSMA) mechanisms and uses clustering techniques. The Breath protocol was designed for control applications. But with the use of CSMA and in the attempt to be adaptive to random conditions, the protocol is aimed at applications that has random possibilities or has



(a) Chemical plant with process control automation

(b) Process control graph

Fig. 2: Scenario description and chart representation of process control variables

continuously varying sampling intervals (time duration between each sensor reading). For process control applications which are known to have precise process models that define transient and steady states along with steady state error, could benefit from the use of schedule based protocols, increasing predictability, and also energy efficiency. In[7], the authors also assume that requirements for TDMA such as having a network manager, are complex requirements. This is in contrast to our protocol, where we assume that these are a common part of the network solution. Thus the two protocols aim at solving different problems. Generic wireless solutions proposed as industrial standards generally have an architecture that consists of a network manager and other powerful middleware. Two popular wireless standards are wirelessHART[8] and ISA100.11a[9]. Our proposed DMA-MAC protocol can be used in combination with both wirelessHART and ISA100.11a, which are based on the 802.15.4 standard.

3. Networked Control System Example

We define an example scenario of a chemical processing plant, employing NCSs for automating a chemical process using sensors and actuators. The example deals with different kinds of sensors including temperature, pressure, and level. The scenario is similar to the one used in one of the case studies for the GinMAC[10] protocol. Figure 2a shows a schematic diagram of the process. We assume that two chemicals A and B are stored in two different tanks (Tank 1 and Tank 2) and then mixed in the reaction vessel. The reaction is continuously monitored and controlled for temperature, pressure, and level using a feedback-based control system. The goal of the system is to supervise the mixing operation. This is done by continuously measuring all parameters (temperature, pressure, and level) via sensors and making necessary control decisions. The mixing procedure starts with letting the chemicals flow to the reaction vessel. During this process, the flow measurement and management is very important. After the flow is completed and the chemical reactions take place, it is important to continuously manage pressure and temperature. The temperature is controlled via heat exchangers in the reaction vessel. Initially, when the chemicals flow and the reaction starts, there is possibly a high rate of change of temperature and pressure in the reaction vessel (assuming flow is constant). This indicates a transient state. After a while, when the flow is completed, and a given temperature is reached, the rate of change in temperature and pressure will be small. This indicates a steady state. For the example in figure 2a, a typical process control graph is illustrated in figure 2b. The graph is plotted for temperature varying with time, continuously monitored and controlled. Similar behaviour is exhibited by the remaining parameters (pressure and level). We have a target value for temperature (set point or reference), and two bounds (upper and lower) between which the temperature may vary, while the system is still considered to be in the steady state.

4. The DMA-MAC Protocol

The DMA-MAC is proposed based on the dual-mode operation of process control as introduced above. It has one mode for the transient state, and one mode for the steady state. This allows for service differentiation, thus facilitating support for the faster data rate in the transient mode, and saving energy by reducing the data rate in the steady mode. We define two separate superframes to represent slot distribution in the two different operation modes. DMA-MAC is entirely based on a TDMA slot mechanism in both superframes except for alert messages. Alert messages handling is explained below. One important part of DMA-MAC operation is the decision of switching the mode of operation. The transient to steady mode switch is done by the sink based on preset characteristics, i.e, when system is continuously in steady mode operation for the past 10s (sensor reading is within steady state intervals). The steady to transient switch is more critical than the former, and is detected and notified by sensors to the sink. We use thresholds to identify the switch from steady to transient. As represented in the process control graph in figure 2b, the lower bound and upper bound define the threshold. When in steady mode operation, the sensors constantly monitor for measurements that go beyond the threshold, and when such an event occurs, the sink is notified. The sink then sends a signal indicating the switch of operational mode. Below, we discuss in detail the two operational modes of DMA-MAC.

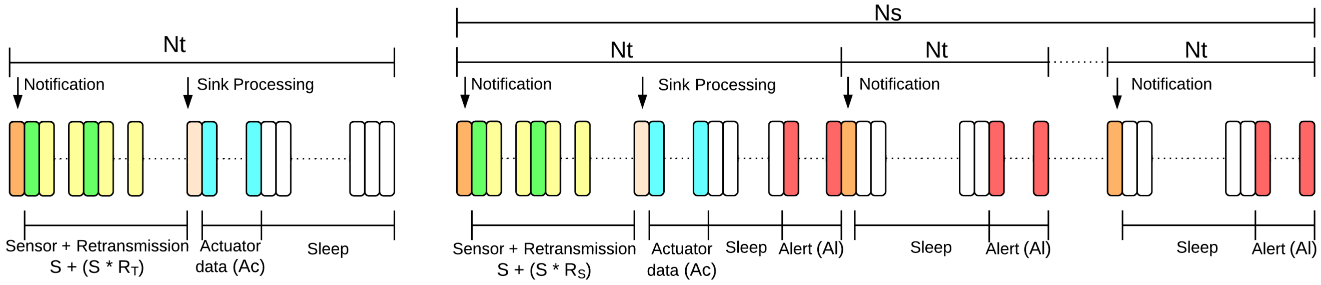
Transient mode operation. The process changes rapidly during transient state, generating large volume of data sampled at high frequency, to trace the rapidly changing process variables (such as temperature). The sensors transmit this data to the sink. Actuators continuously act on this data. We have a short superframe with Nt (t for transient) slots and smaller sleep duration compared to steady mode operation, to increase data reliability. In general, transient mode operation is data-intensive and hence also energy intensive. The transient mode operation superframe is based on the GinMAC[5] superframe with similar structure except for the placement of actuator slots. The superframe structure for transient mode operation is illustrated in figure 3a and consists of:

- 1 notification slot for information dissemination from the sink to all sensors and actuators. This includes time synchronization, switching of operational mode, change of thresholds, and other control data. With the switch notification, all nodes are notified to switch in the next superframe.
- S slots for sensor data transmission towards the sink.
- $S * R_T$ re-transmission slots for increased reliability. Here R_T is the number of re-transmissions per sensor data. By setting the number of re-transmissions for transient and steady mode operation, we can have further control the tradeoff between energy conservation and reliability.
- 1 slot for information processing at the sink.
- Ac slots for actuator data transmission from the sink.
- The remaining slots out of the Nt slots are used as sleep slots.

Steady mode operation. The steady mode operation in DMA-MAC is defined to operate during the steady state of the process. During the steady state, the data-rate requirement of the controller is low, and thus we also keep the communication of the sensed data low to save energy. The superframe structure for steady mode operation has Ns (S for steady) slots and is designed to be a multiple of Nt (number of slots in transient mode). This is to ensure that the maximum delay between the detection of a threshold violation and change of superframes is two transient superframe in length. The superframe structure is shown in figure 3b, and consists of:

- Ns/Nt notification slots for information dissemination, similar to transient mode operation. With the switch notification, all nodes are notified to switch in the next Nt part of the steady mode superframe.
- S slots for sensor data transmission towards the sink.
- $S * R_S$ re-transmission slots, here R_S is based on the link conditions, similar to re-transmission slots in GinMAC. Also, here R_S is the number of re-transmissions per sensor data set for steady mode operation.
- 1 slot for information processing at the sink.
- Ac slots for actuator data transmission from the sink.
- $Al * (Ns/Nt)$ slots for alerting the sink of a need for switch of operational modes from steady to transient. After every Nt slots in the steady mode superframe, we repeat the notification slots, and alert slots.
- The remaining slots out of the Ns slots are used as sleep slots.

Alert slots are used to indicate the switch of process state from steady to transient state. The switch is detected by the sensors based on thresholds determined from the process model. The sending procedure for alert messages does



(a) Transient mode superframe

(b) Steady mode superframe

Fig. 3: Superframes for the two modes of the DMA-MAC protocol

not adhere to TDMA, but is based on a special procedure. During the alert message sending slots, mainly the parent nodes are awake to relay any information coming from their children. All leaf nodes are at sleep, unless they have alert messages to be sent. Alert messages are usually short messages and two (one duplicate) of these messages are sent in the same slot to increase the probability of receiving the alert message. Any of the child nodes could send an alert message to its parent in alert slots. Thus we could have a possibility where two child nodes of the same parent have alert messages created and need to send them in the same slot. To solve the possible collision at the relay node, we introduce a random delay in the interval [0..slotDuration/n] before sending the first alert message, and then again a random delay within the same range [0..slotDuration/n] as the previous one before sending the next alert message. We use $n \geq 3$, to ensure that the two alert messages can be accommodated in the same slot. Thus, we attempt to increase the probability of at least one of the alert messages reaching the sink, and the operational mode switch being triggered. An implicit ACK mechanism is used, where the node that generated alert message listens to the channel in the next slot to make sure that the alert message is relayed towards the sink by the parent node. We can also consider the addition of CSMA for alert message transmission, to further reduce collision.

The *Sink* sets the threshold depending on the process model given as input by the process model designers. The sink is also responsible for detecting the change in states from transient to steady, make the necessary switch in operational modes, and inform all sensors and actuators under its management. Timing synchronization is handled by the sink, which is a basic requirement for having TDMA operation. *Time Synchronization* information is sent in the information notification slot every λ frames. It is sent in the beginning of the frame for transient mode operation. For steady mode operation it is sent in the beginning of the superframe and then after every Nt slots. We assume a powerful sink and that the message sent by the sink is received by all nodes in the network. We use acknowledgements (ACK) for all packets sent by the sensors. For packets sent by sink, no ACK is used. The ACK packets are sent within the same time slot after the data packets. On failure of the data or ACK packets, re-transmission slots are used.

5. Initial Evaluation of Energy consumption

In this section, we describe the energy model for both GinMAC and DMA-MAC, and provide a first comparison between them. We assume that the transient mode superframe and the GinMAC superframe are of the same length to have similar performance for transient state. We use δ to denote the slot duration (e.g. 10ms). The main radio states being TX - Transmission, RX - Receiving, and SLEEP - Sleeping. We define $P_{RadioState}$ as the power consumed by the radio in the defined radio state. We denote energy consumed in a given radio state by $E_{radioState}$. $E_{Protocol}$ is the total energy spent in a single superframe by a given protocol (having one type of superframe structure). $E_{SLEEP}^{superFrameType}$ is used to describe the total sleep energy spent in one superframe of a given superframe type. Similarly, $E_{protocol}^{superFrameType}$ is used to describe the total energy spent in one superframe of a given superframe type of the respective protocol. We define α as the total number of transient superframes for a given time duration T. $E_{protocol}(\alpha)$ denotes the total energy spent for α superframes using the given protocol. p defines the probability of transient superframes appearing and $(1-p)$ defines the probability of steady superframes appearing. We present this analysis for one hop networks, and we assume in the case of alerting, that the switch always happens in the last alert duration of the superframe. We assume worst case for re-transmissions, i.e., that all retransmissions are used. A more detailed calculation of energy consumption would also take into account the switching power and time. For simplicity, we neglect these, and the computation of energy spent by sink which is wire powered is omitted. Based on the assumptions, we have:

$$E_{TX} = P_{TX} * \delta,\ E_{RX} = P_{RX} * \delta,\ E_{SLEEP} = P_{SLEEP} * \delta,\quad \text{for worst-case analysis we assume that the entire slot is used.}$$

$$E_{SLEEP}^{Transient} = E_{SLEEP} * (Nt - 1 - S - (S * R_T) - Ac),\quad \text{subtraction of 1 is for notification slot.}$$

Alert is considered only once in a steady superframe, after which transient superframe starts and hence:

$$E_{SLEEP}^{Steady} = E_{SLEEP} * (Ns - (1 * (Nt/Ns)) - S - (S * R_S) - Ac - Al),\quad (1 * (Nt/Ns) \text{ is for notification slots} \tag{1}$$

$$E_{GinMAC}(\alpha) = \alpha * (E_{RX} * 1 + E_{TX} * S + E_{TX} * (S * R_T) + E_{RX} * Ac + E_{SLEEP}^{Transient}) \text{ for } \alpha \text{ superframes in time T.} \tag{2}$$

$$E_{DMA-MAC}^{Transient} = E_{GinMAC}$$

$$E_{DMA-MAC}^{Steady} = E_{RX} * (Nt/Ns) + E_{TX} * S + E_{TX} * (S * R_S) + E_{RX} * Ac + E_{TX} * Al + E_{SLEEP}^{Steady} \tag{3}$$

$$E_{DMA-MAC}(\alpha) = p * \alpha * E_{DMA-MAC}^{Transient} + (1-p) * \frac{\alpha}{Ns/Nt} * E_{DMA-MAC}^{Steady} \tag{4}$$

Table 1 summarizes the relative energy consumption of DMA-MAC and GinMAC. For conditions where we have high number of steady state superframes with $p < 0.1$ for transient superframe to appear during operation, the DMA-MAC has far better energy performance than GinMAC. For other possibilities when the process state has $p < 0.5$ for the next frame to be transient frame, DMA-MAC has still lower consumption of energy. For process models where transient state dominates through the process duration it is advisable to use single-mode protocols like GinMAC especially to prevent the overhead, and this is represented by the $p = 1$ case.

Condition	Energy consumption
$p < 0.1$	$E_{DMA-MAC}(\alpha) << E_{GinMAC}(\alpha)$
$p < 0.5$	$E_{DMA-MAC}(\alpha) < E_{GinMAC}(\alpha)$
$p = 1$	$E_{DMA-MAC}(\alpha) = E_{GinMAC}(\alpha)$

Table 1: Results summary

On a simpler note, the main advantage of energy saving is in the steady state of DMA-MAC as shown for a small comparison in Figure 4. Here we show the superframe of normal steady mode (SF), and the steady mode superframe (SF^*) where a switch happens once for a time duration T. Assuming that energy spent on sleep and alert are negligible for a given time T (with α superframes), the energy consumed by DMA-MAC steady state is approximately equal to the energy consumed by GinMAC divided by the multiple (Ns/Nt), we have:

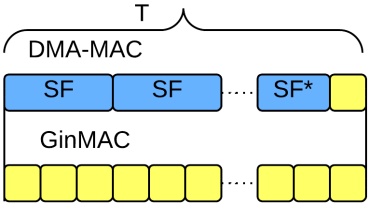


Fig. 4: Steady state

$$E^{Steady}_{DMA-MAC}(\alpha) \simeq \frac{\alpha}{Ns/Nt} * E_{GinMAC} \quad \text{or from eqn.(2):} \quad E^{Steady}_{DMA-MAC}(\alpha) \simeq \frac{E_{GinMAC}(\alpha)}{Ns/Nt} \tag{5}$$

6. Conclusion

In this article, we have proposed a WSAN protocol DMA-MAC that is specifically designed for process control applications and requirements. Also, we consider the system dynamics for the process control applications. The system dynamics for process control include two prominent states: steady state and transient state. We have based the protocol design on reflecting the system dynamics states with dual-mode operation. With dual-mode operation, we aim to obtain a better energy efficiency preserving the required reliability and delay requirements. By construction and initial performance analysis, we see that DMA-MAC is energy efficient for certain process conditions that we discussed in the energy consumption evaluation section. An important part of future work is to conduct a more elaborate performance analysis based on other important metrics for process control applications including delay, packet error rate, throughput, and reliability. Furthermore, we will extend the energy evaluation to consider multi-hop communication. A practical evaluation of the proposed design is also planned both in terms of validating that the new protocol is better in practice, and also in terms of figuring out whether there are elements of the design which, in practice is difficult to implement.

References

1. I. F. Akyildiz, I. H. Kasimoglu, Wireless sensor and actor networks: research challenges, Ad Hoc Networks 2 (4) (2004) 351–367.
2. J. P. Hespanha, P. Naghshtabrizi, Y. Xu, A survey of recent results in networked control systems, Vol. 95, 2007, pp. 138–162.
3. G. Zhao, Wireless sensor networks for industrial process monitoring and control: A survey., Network Protocols & Algorithms 3 (1) (2011) 806–838.
4. A. Bachir, M. Dohler, T. Watteyne, K. Leung, MAC essentials for wireless sensor networks, IEEE Communications Surveys Tutorials 12 (2) (2010) 222–248.
5. P. Suriyachai, J. Brown, U. Roedig, Time-critical data delivery in wireless sensor networks, in: Proceedings of DCOSS, 2010, pp. 216–229.
6. T. O Donovan, J. Brown, U. Roedig, C. J. Sreenan, J. do O, A. Dunkels, A. Klein, J. Silva, V. Vassiliou, L. Wolf, GINSENG: Performance control in wireless sensor networks, in: SECON, 2010, pp. 1–3.
7. P. Park, C. Fischione, A. Bonivento, K. Johansson, A. Sangiovanni-Vincent, Breath: An adaptive protocol for industrial control applications using wireless sensor networks, IEEE Transactions on Mobile Computing 10 (6) (2011) 821–838.
8. J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, Wirelesshart: Applying wireless technology in real-time industrial process control, in: IEEE Real-Time and Embedded Technology and Applications Symposium, 2008, 2008, pp. 377–386.
9. Draft standard ISA100. 11a, in: Internal working draft, International Society of Automation, 2008.
10. F. Busching, W. Pottner, D. Brokelmann, G. Von Zengen, R. Hartung, K. Hinz, L. Wolf, A demonstrator of the ginseng-approach to performance and closed loop control in wsns, in: Ninth International Conference on Networked Sensing Systems (INSS), 2012, pp. 1–2.

Chapter 8

Simulation-based Evaluation of DMAMAC: A Dual-mode Adaptive MAC Protocol for Process Control

Simulation-based Evaluation of DMAMAC - A Dual-Mode Adaptive MAC Protocol for Process Control

A. Ajith Kumar S.
Bergen University College,
Norway
University of Agder, Norway
aaks@hib.no

Lars M. Kristensen
Bergen University College
Bergen, Norway
lmkr@hib.no

Knut Ovsthus
Bergen University College
Bergen, Norway
kovs@hib.no

ABSTRACT

Control systems automation is widely used in many industrial domains and have strong requirements on delay, throughput, robustness, and reliability. In the domain of networked control systems, the medium of communication is increasingly involving wireless communication along-side conventional wired communication. Issues ranging from energy efficiency and reliability to low-bandwidth have to be addressed to enable the transition to increased use of wireless communication. In earlier work, we have proposed the Dual-Mode Adaptive MAC (DMAMAC) protocol relying on a combination of Time Division Multiple Access (TDMA) and Carrier Sense Multiple Access (CSMA). The DMAMAC protocol is able to dynamically adapt to the two main states found in process control: the steady state and the transient state. Key requirements to the DMAMAC protocol are energy efficiency, low probability of state-switch failures, and a low state-switch delay.

The contribution of this paper is a comprehensive simulation-based evaluation of the original DMAMAC protocol along with the evaluation of a new pure TDMA-based variant of the DMAMAC protocol. Our results show that for processes where the steady state dominates, both variants of the DMAMAC protocol can reduce energy consumption by up to 45% in comparison to the closely related single-mode GinMAC protocol. Among the two variants of DMAMAC, the pure TDMA-based variant has the better energy efficiency and higher reliability. The simulation results also show that the hybrid TDMA-CSMA variant of the DMAMAC protocol has a probability of less than 0.3% for a state-switch failure in a given MAC superframe. The simulation study has impacted the design of the DMAMAC protocol by providing insights that have led to design changes in the originally proposed DMAMAC protocol in order to further reduce the state-switch delay between the steady and the transient state.

Keywords

Wireless Sensor Actuator Networks, Process Automation, Process Monitoring and Control, Medium Access Control, MAC protocols, Energy efficiency, Reliability

1. INTRODUCTION

A Wireless Sensor Actuator Network (WSAN) [1] consists of sensors and actuators that use radios to send, relay, and receive information. WSANs are used across multiple application domains, including process and factory automation. The advantage of WSANs lies in reducing operating costs, the size of the devices used, and in increasing automation. Feedback-based control-loop automation is one of the main applications, and control systems that use wired or wireless solutions for data transfer are known as Networked Control Systems (NCSs) [4]. A general control system consists of a reference input, plant output, sensors, actuators, and a controller providing control input as shown in Fig. 1. Previously, NCSs used only wired communication due to the high reliability, low delay, and high bandwidth. Now, NCSs are adopting wireless communication co-existing with wired solutions. Given the fact that nodes in WSAN are generally battery powered, energy efficiency is an important requirement in addition to the real-time requirements.

In this paper, we focus on NCS based on wireless communication designed for process control applications. Processes are generally modelled using mathematical process models, which represent the characteristic features of the process. The process model also describes the change in physical quantities (e.g. temperature, pressure, and level) over time. In process control, the process being controlled could be in one of two states at a given time: *steady* and *transient*. In the transient state, the process changes rapidly, i.e., the physical quantities change rapidly. For the process control to operate properly, the sensors need to communicate the rapidly changing dynamics of the process (in transient state) to the sink (gateway), which then aggregates this data

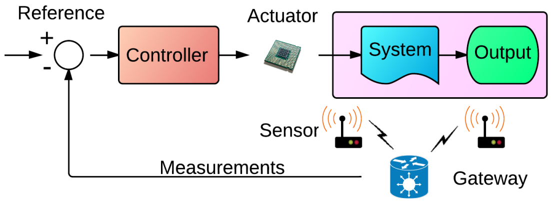


Figure 1: Control loop with wireless communication

and forwards processed control data to the actuators to act on this data. With time, the process starts stabilizing and the rate of change decreases. The process is then said to be in steady state. In steady state, the physical quantity is either constant or has minimal change within a given threshold interval. This threshold interval is also used to detect the need for a switch from steady to transient state. When the sensors measure values outside of the threshold interval, it notifies the sink to initiate a state-switch. This switch from steady to transient can be critical depending on the application requirements.

In [13], we proposed the Dual-Mode Adaptive Medium Access Control protocol (DMAMAC), designed to have two operational modes to cater for the two process control states. The protocol has a transient mode that supports the transient state, and a steady mode to support the steady state. The transient mode has data communication at a higher rate relative to the steady state. We designed DMAMAC for applications where steady state dominates the process operation. Generally, wireless sensor nodes are battery powered and the transceiver consumes more energy than the micro-controller and other parts [3]. Thus, the reduced communication in the steady mode results in energy savings. The transient mode still has energy consumption similar to single mode protocols. A further benefit of the DMAMAC protocol is that it has reduced interference to other operations in the vicinity due to a low duty cycle in the steady state. Previously, in [13] we proposed a hybrid protocol based on a combination of Time Division Multiple Access (TDMA) and Carrier Sense Multiple Access (CSMA). In this article, we simulate this protocol to study the performance in comparison to the GinMAC protocol [15] which is a well-established protocol for industrial monitoring and control. In addition, we develop and evaluate the performance of a new pure TDMA-based variant of DMAMAC (DMAMAC-TDMA) and compare it to the original DMAMAC protocol (DMAMAC-Hybrid) from [13].

The rest of the paper is organized as follows. Section 2 briefly introduces the design of the DMAMAC protocol and the related GinMAC protocol [15] which will serve as a baseline for comparison and design. The simulation setup based on the scenario in [13] is discussed in Sect. 3, along with the MiXiM [6] simulation model used. Both variants of the DMAMAC protocol are evaluated and compared with the GinMAC [15] protocol in Sect. 4. In Sect. 5, we sum up the conclusions and discuss future work. In the rest of the paper we use DMAMAC to refer to our protocol in general and append either Hybrid or TDMA to designate the specific variant of the protocol in question. We assume that the reader is familiar with the basic concepts of MAC protocols.

2. RELATED WORK AND DMAMAC

In this section we briefly discuss related work, and describe the DMAMAC protocol with focus on the changes made to the previous version [13]. In related work, we focus specifically on the TDMA-based GinMAC protocol [15] as the design of DMAMAC is rooted in this protocol. In addition to GinMAC, there are several other related MAC protocols. Z-MAC [11] is a hybrid protocol using both CSMA and TDMA. Z-MAC is a distributed protocol with two-phase operation. In the setup phase the nodes make a list of their two-hop neighbors and then locally decide on a time slot such that no two nodes select the same time slot. Z-MAC uses CSMA for low-contention and TDMA for high contention periods. The DMAMAC protocol is based on offline scheduling similar to GinMAC. This means that the entire time-slot is pre-planned and thus differs in operation from Z-MAC. WirelesHART [14] is the wireless successor of HART proposed for process monitoring applications. WirelessHART is a framework, with a combination of protocols for different functions. TSMP [10] is the MAC protocol proposed for wirelessHART, but it mainly handles time synchronization. The routing and slot allocation is handled by a separate unit called the network manager. The DMAMAC protocol can be used within the wirelessHART framework as a MAC protocol.

2.1 The GinMAC Protocol

GinMAC [15] was developed as part of the GINSENG [7] project with requirements including reliable and timely delivery of data. The GinMAC protocol design is based on a network having a tree topology. Along with satisfying real-time requirements, GinMAC also addresses energy efficiency via efficient duty cycling. The GinMAC superframe forms the basis of the transient mode superframe in the DMAMAC protocol. The GinMAC protocol has the following main characteristic features.

Off-line Dimensioning. The network deployment is pre-planned based on application requirements, and scheduling decisions are made offline. A TDMA schedule is created with a given superframe length. This frame is then divided into three types of slots: *basic*, *additional*, and *unused.* The *basic* slots are for regular sensor data transfer, the *additional* slots are used to increase reliability, and the *unused* slots (or sleep slots) are used to achieve a low duty cycle. *Basic* slots are defined for both sensor and actuator data. *Configuration* commands include control data such as time synchronization. The sensor data is sent from the sensors to the sink, and the actuator data is sent from sink to the actuators. Given the tree topology, nodes having children need to have basic and additional slots allocated for its children as well.

Exclusive-TDMA. GinMAC is designed such that data transmission of maximum length and acknowledgement is accommodated within the same slot. GinMAC uses exclusive TDMA with no slot re-use across nodes. The GinMAC protocol has been designed for a sink that can manage a maximum of 25 nodes [15].

Delay Conform Reliability Control. Additional slots are used to ensure packet delivery thus increasing reliability. Prior to deployment, measurements are to be performed in the deployment area to assess the channel characteristics and calculate the worst-case link reliability. The number of additional slots used is based on the calculated worst-case link reliability.

2.2 The DMAMAC Protocol

The hybrid variant of the DMAMAC protocol was initially proposed in [13], and has been further refined in the context of this paper to suit the goals of the protocol design and for improvements in performance. In addition, we propose a pure TDMA-based variant of the protocol. The detailed description of the DMAMAC-Hybrid protocol can be found in [13]. In this paper, we discuss in brief the two operational modes of DMAMAC, the changes made with respect to the original Hybrid variant, and the pure TDMA-variant. Also, the delay from the time instance the state-switch is identified until the sink is notified is discussed. We rely on a tree topology similar to that of GinMAC [15], and DMAMAC also follows the offline scheduling and reliability mechanisms of GinMAC. The network architecture, for which the DMAMAC protocol is designed, consists of a sink node, sensor nodes, and actuator nodes. The sensor/actuator nodes are ranked according to their position in the tree topology, with nodes closest to the sink having the lowest rank. The sink node is responsible for managing the entire network, and is assumed to be wire powered and computationally powerful. Below, we list the key design considerations and assumptions related to the design of the DMAMAC protocol:

1. The protocol is designed for applications where the steady state is dominant.
2. For multi-variable process models, we design the transient mode operation to continue until the slowest of the inputs reaches its steady state.
3. The setting of thresholds for the sensors to detect the state-switch is assumed to be based on the underlying process model.
4. The sink is assumed to be able to reach all nodes in one hop. Notification messages from the sink are sent in one-way communication without ACK in one slot to all nodes in the network, and include control data such as state-switch data. Since the sink is wire powered, it is reasonable to assume that it can afford to have longer radio range.
5. A small amount of packet failure is tolerated by the control system. Model Predictive Control (MPC) [8] or network-aware control systems can be used to compensate for the possible packet losses.
6. Static network topology: no addition or removal of nodes during operation. In case of a topology change, schedules are recomputed accordingly.
7. A single slot accommodates both DATA and ACK packets.

2.2.1 *Transient mode*

The process changes rapidly during the transient state, and the transient mode is designed to meet the data requirements of this state. The transient superframe of DMAMAC is shown in Fig. 2. It is similar to the GinMAC superframe, but differs in the actuator data slot positions. The transient superframe is of length N_t (t for transient) slots and has smaller sleep duration compared to steady mode operation to increase data reliability. Both DMAMAC-Hybrid and DMAMAC-TDMA have the same transient superframe structure.

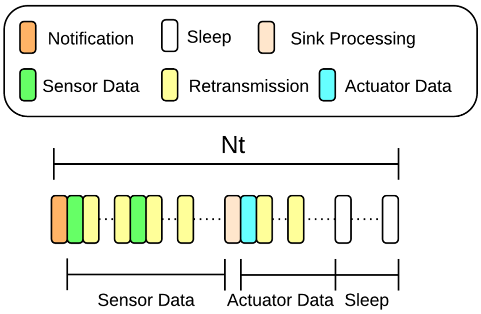


Figure 2: Transient mode superframe

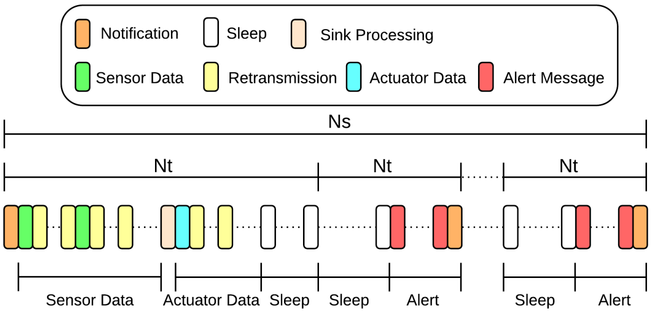


Figure 3: Steady mode superframe

2.2.2 *Steady mode*

During the steady state, the data-rate requirements of the controller is low, thus we also keep the communication of the sensed data low to save energy. The superframe structure shown in Fig. 3 for steady mode operation has N_s (s for steady) slots and is designed to be a multiple of N_t (number of slots in transient mode). Both DMAMAC-Hybrid and DMAMAC-TDMA have the same steady superframe structure, except for their alert slot scheduling which is discussed later with alert messages.

2.2.3 *State-switch delay*

The state-switch delay is the time interval between identification of a threshold breach by the sensor and the state-switch happening in the network, i.e., a change from the use of steady mode superframe to transient mode superframe. The steady superframe considered in this paper has been changed with respect to our previous proposal [13]. In the previous version of the Hybrid variant of the protocol, the alert slots were placed towards the end of the data communication part or the N_t long sleep parts. The notification slots were placed in the beginning of each N_t long part of the steady state superframe. Thus when an alert is detected, a notification is sent to the entire network in the next N_t part, the one after the N_t part where the alert is detected by a sensor, and is notified to the sink first. However, the change of superframe would happen in the N_t part that follows the N_t part with notification. This effectively implies a minimum state-switch delay of $2 * N_t$ in the earlier version. With the current version, the minimum state-switch delay is reduced to N_t. Based on the current superframe structure, the state-switch delay is either two

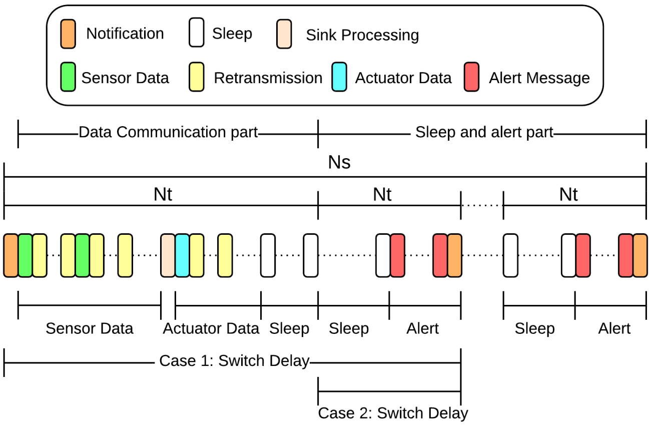


Figure 4: The two state-switch delay cases

transient superframes (case 1) or one transient superframe (case 2) as illustrated in Fig. 4. The two cases are:

- Case 1: occurs if the alert is generated in the first part of the steady superframe (the data communication part). In this case, the state-switch happens after the second N_t part. Since the data is sent already in the first part, the real delay between two consecutive data communication part is still one transient superframe length.
- Case 2: is when the alert is generated in one of the sleep parts of the steady superframe, which will then have a state-switch delay of one transient superframe length.

Case 1 can be eliminated by adding alert and notification slots at the end of the data communication part. For case 2, we can further decompose the second part of steady superframe into a half or a quarter of the transient superframe length each ending with alert and notification slots. It should be noted that these two cases represent the state-switch delay when there is no packet loss. The packet loss conditions where the state-switch attempt fails are discussed in Sect. 4.

2.2.4 Alert Messages

With respect to the previous design [13] we have updated the alert message part for the DMAMAC protocol. Now we have two methods to cater for alert messages. The first one is the existing alert message sending method proposed earlier for DMAMAC-Hybrid. The second is a method for the TDMA-variant of the protocol with alert message handling, where each node has a separate alert transmit slot in the alert period. The two methods are described below.

DMAMAC-Hybrid alert method. As mentioned above, in [13] we proposed sending two alert messages in one slot to increase the probability of the alert messages reaching the next hop. The design consisted of one alert slot per rank in the network topology. Our initial simulation results showed that sending one alert message along with a Clear Channel Assessment (CCA) has a better performance than the former method. The main idea here is to have minimal number of alert slots whilst maintaining low probability ($< 1\%$) of state-switch failures. Given the network topology, we have one alert slot for each rank (level in the tree topology). We still maintain a random delay before sending the alert message but now with a duration within the interval [0..(*slotDuration − (Maximum time required to send alert message))*]. The random delay along with CCA reduces the collisions (within the network), and thus reduces state-switch failures. Given the superframe structure collision is only possible in alert slots. Thus, collision is known to be a result of two nodes (at least) transmitting alert packets simultaneously.

DMAMAC-TDMA alert method. In DMAMAC-TDMA, each node has its own alert slot. This ensures no collision and reduces the possibility of switch failures. Switch failure is still possible due to packet loss on the wireless channel. The parent nodes send one alert based on either its own alert, or forwarding of an alert received from one of its children. Thus, the total number of alert slots is equal to the number of sensor nodes in the network. This method provides better reliability for alert messages to facilitate the state-switch from steady to transient which can be critical under consideration for the application.

2.3 Parameters and Design Alternatives

The DMAMAC protocol has several parameters that can be adjusted in accordance to the various trade-offs. These are listed below along with design alternatives:

1. The length of the sleep parts (currently N_t) in the steady superframe can be varied, which determines the maximum delay in state-switch, and also impacts energy savings.
2. The number of transient superframe length (N_t) parts to be used in the steady superframe, impacting energy savings.
3. The re-transmission count can be varied, impacting reliability. Re-transmission can be added into the alert slots as well.
4. Aggregation of data packets at the parent nodes. The order of the slots would then be: first the slots for the parent data packets and then slots for aggregated data from child nodes.
5. The number of alert slots could be increased to two slots for each rank for the Hybrid variant. This would increase the reliability of alert.

3. SIMULATION MODEL AND SETUP

For the evaluation of the DMAMAC protocol, we perform simulation-based analysis. Based on the scenario considered in [13], a topology is designed for the simulation and is shown in Fig. 5. The topology has a total of 26 nodes including the sink. The numbers in Fig. 5 refer to node numbers, and numbers prefixed with ”R” refer to rank in the tree topology. The topology can be configured in different ways. We use this representative topology to explore the key aspects of the DMAMAC protocol which are data communication, alert messages, multiple hops, and the possibility of collision (the latter only for the Hybrid variant). Different configurations of each protocol variant (TDMA and Hybrid) are used for evaluating the protocol, and the performance of the protocol on the considered topology is discussed later. The topology consists of 19 sensor nodes, 6 actuator nodes, and 1 sink node. The tree topology has 3 ranks of node placements, with the most distant leaf node being 3 hops away from the sink. The nodes closest to the sink have the highest load in the network. Based on the setup, node 3 will have the highest data load among the three nodes with the highest rank. Given this load distribution and equal initial battery level on all nodes, node 3 is expected to run out of energy before any other node in the network. Thus for our experiment, we calculate network lifetime based on time to first node death.

Simulation is done using MiXiM [6] which is an OMNeT++ [16] based modeling framework designed for simulating wireless networks. We simulate DMAMAC in different configurations mainly with different probabilities of transient superframes appearing. We compare DMAMAC with the performance of GinMAC in a similar configuration. The simulation parameters are listed in Table 1. We use the radio parameters from the CC2420 datasheet [5], a radio unit frequently used in sensor and actuator nodes. The current consumed by the radio in different states Receive (RX), Sleep (Sleep), Transmit (TX), setup currents, and switch currents is defined using the data obtained from the CC2420 datasheets. The time used to switch between radio states is also obtained from the CC2420 datasheets. We evaluate the protocol under ideal channel conditions.

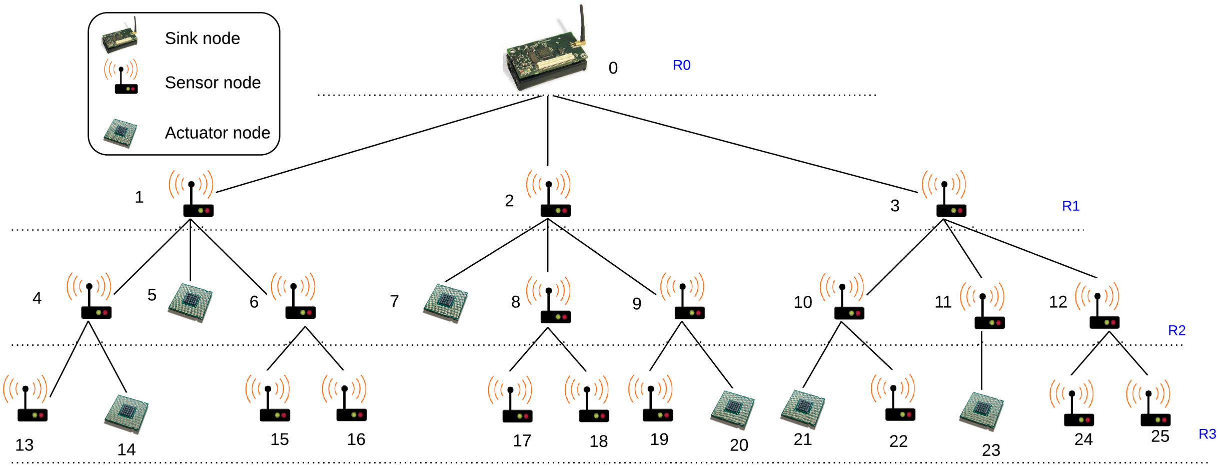


Figure 5: The logical topology for the simulation analysis

Parameter	Values
Number of Nodes	25 Nodes
Network dimensions(metres)	160 * 160
Simulation duration	1350 seconds
Transient Superframe length	1.5 seconds
Distance between nodes	15 to 25 meters
Number of rounds	900
MAC Protocols	DMAMAC-Hybrid DMAMAC-TDMA GinMAC
Radio Module	CC2420
Simulation repetition	100
State switch probability	10, 50
Superframe Ratio	2x, 3x, 4x
Data Packet size	44 bytes
Sink Packet size	11 bytes
ACK, Alert Packet size	11 bytes

Table 1: Simulation parameters

An overview of the MiXiM simulation model used is shown in Fig. 6. In the MiXiM platform, we have created a project that contains C++ files, a NED package, XML files, packet description files, and a input parameters file. The C++ files are used to describe the MAC protocol for both the regular nodes and the sink node. The NED language in OMNeT++ is used to describe the topology, the connection between different modules, and the hardware characteristics. Using NED files, we describe the regular nodes and the sink node along with their network interface cards (NIC). XML files are used as input files in the C++ files. The config XML file is for defining the path loss model used.

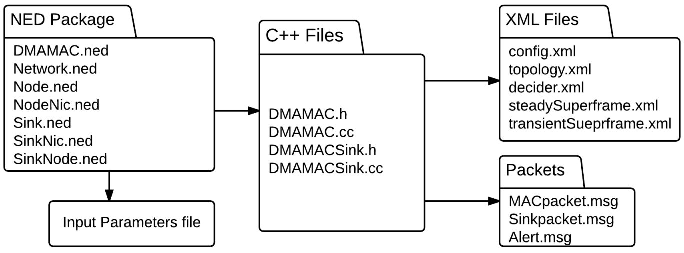


Figure 6: Overall structure of the MiXiM model

The decider XML file describes the decider characteristics for the simulation. In this case we use the CC2420 decider for signal evaluation and demodulation [6]. Topology XML files gives the overall topology of the network describing the interconnection between different nodes. The different packets are: the regular MAC packet, Sink MAC packet, and the MAC packet used for Alert messages. These are all defined in separate files. The input parameters file defines values for the DMAMAC protocol parameters that can be varied to obtain different configurations. We refer to [6] and [16] for detailed information about designing simulation models in MiXiM.

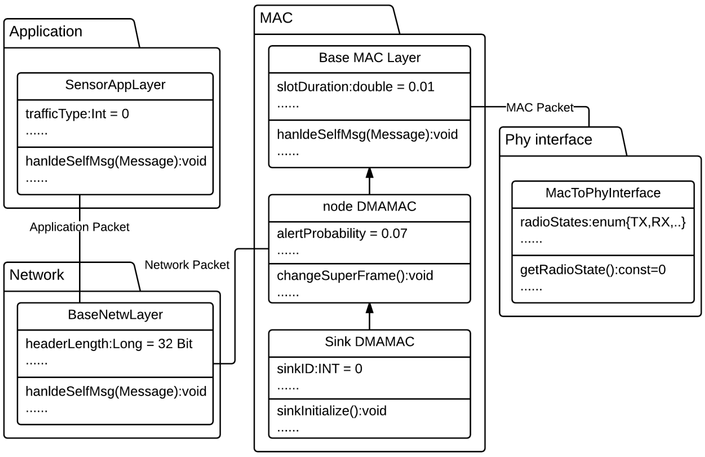


Figure 7: Overall architecture of the DMAMAC implementation

The overall architecture of the DMAMAC protocol implementation in C++ is shown in Fig. 7. We implemented the DMAMAC protocol for the nodes based on the inheritance from the BaseMacLayer from the MiXiM library. This is further inherited by the DMAMAC protocol for the sink (differs from nodes). We rely on the application and network layer from the MiXiM library defined in the input parameters file. The MAC protocol accesses the radio/physical layer via the interface provided in MiXiM, and is used to switch between different radio states, mainly transmit (TX), receive (RX), and sleep (SLEEP).

The scheduling is done offline, and is given as input XML

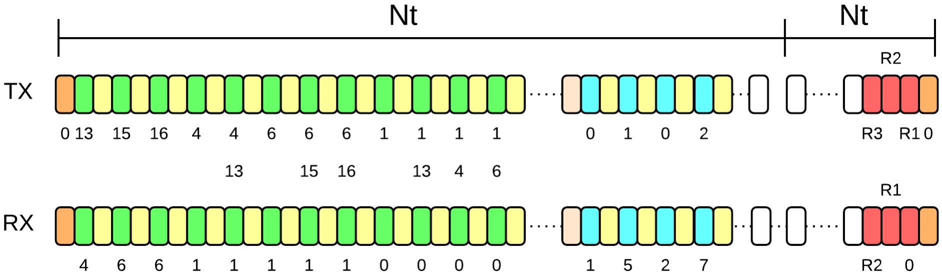


Figure 8: Slot schedule

files separately for transient and steady superframe. A fragment of this is illustrated in Fig. 8. TX represents the transmitting node number and RX represents the receiving node number. The numbers prefixed with R represent the alert slot for a given rank in the tree topology, with leaf node rank represented by R3, and the lowest rank represented by R1. The sink has node identification 0. Notification messages are received by all nodes and hence there is no representation for the notification slot in RX. The second row of the numbers below the TX slots represents the child node, the data of which the parent transmits in the given slot. The sink node initiates the network wide state-switch, by sending a notification message to all nodes. The nodes then change the superframe upon receiving the state-switch notification using the appropriate XML input file describing the slot allocation schedule for the new state.

Configurations. We use three main configurations for the simulation study. They differ from each other on the probability of the transient state appearing in a given duration of time. The main configurations have transient state probability 10%, 50% and 100%. The state probability with 100% represents the GinMAC protocol and the other two represent the DMAMAC protocol. The 10% transient superframes configuration represents processes that are stable through most of the process execution, 50% transient superframes appear in relatively less stable processes. These are further evaluated for multiple configurations of superframe length ratio (transient to steady superframe). Configurations with 2x represent a steady superframe which is twice the length of transient superframe, 3x three times, and 4x four times. The ratio depends on the data rate required by the application in the steady state, the lower the data rate requirement, the higher the ratio can be used. All the considered configurations are listed in Table 2. The alert probability is the probability with which each sensor (not actuator) node generates an alert message. The alert probability was obtained using preliminary simulations in order to obtain frame distributions that correspond to state probability of 10% and 50%.

	[%]	Alert Probability		
Config.	Transient	DMAMAC		Superframe
	Probability	Hybrid	TDMA	Multiplier
P-10-2x	10	1.00	1.20	2x
P-10-3x	10	0.80	1.00	3x
P-10-4x	10	0.70	0.70	4x
P-50-2x	50	11.80	11.00	2x
P-50-3x	50	9.10	10.50	3x
P-50-4x	50	8.20	10.00	4x
GinMAC	100	0	0	1x

Table 2: Parameter configurations considered for the evaluation

4. PERFORMANCE EVALUATION

We begin with the discussion of frame distribution, which is the ratio of transient superframes to steady superframes. This ratio in turn determines the ratio betwen transient states and steady states of the network within the simulation. Then, we discuss the energy performance of the two variants of the DMAMAC protocol in comparison with GinMAC protocol via total energy consumption and network lifetime metrics. Further, given the random nature of state-switch requests in DMAMAC-Hybrid, and the CSMA-based method of transmission, alert messages could suffer collision resulting in state-switch failures, which is critical to the operation of the control system. Thus, state-switch failure is discussed in detail.

4.1 Frame Distribution

We investigate the reduction in energy consumption when the fraction of transient superframe is below 10% and also at 50% to give a broader evaluation. In Table 3, we detail the frame distributions considered. For the given simulation time, we have set the alert probability and the transient state probability such that it yields a transient state percentage corresponding to our desired configuration. Table 3 lists the obtained frame distribution across 100 runs. GinMAC has a total of 900 superframes for the simulation duration, and the transient superframes in DMAMAC are measured relative to 900 possible transient superframes for the same duration. The Average column gives the average number of superframes across the 100 runs. The alert probability of each node is independent to that of the other nodes. It was therefore required to estimate the correct alert probability and state probability combination in order to obtain the desired frame distribution. We conducted several simulation runs to obtain proper alert probability. Note that the transient state probability and the alert probability are two different probabilities. Transient state probability applies only for the state-switch from transient to steady which is

	Transient Superframes			
	DMAMAC-Hybrid		DMAMAC-TDMA	
Config.	Average	[%] of total	Average	[%] of total
P-10-2x	87.94	9.77	88.37	9.82
P-10-3x	88.36	9.82	88.28	9.81
P-10-4x	88.28	9.81	88.43	9.83
P-50-2x	441.41	49.05	441.61	48.96
P-50-3x	446.26	49.58	446.04	49.56
P-50-4x	449.03	49.89	448.33	49.81

Table 3: Frame distribution across 100 runs

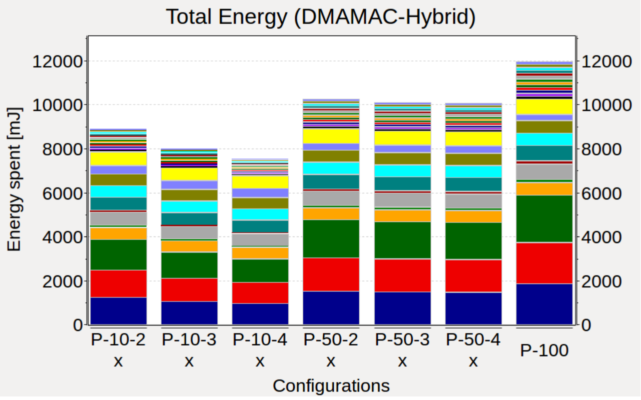


Figure 9: Energy consumption DMAMAC-Hybrid

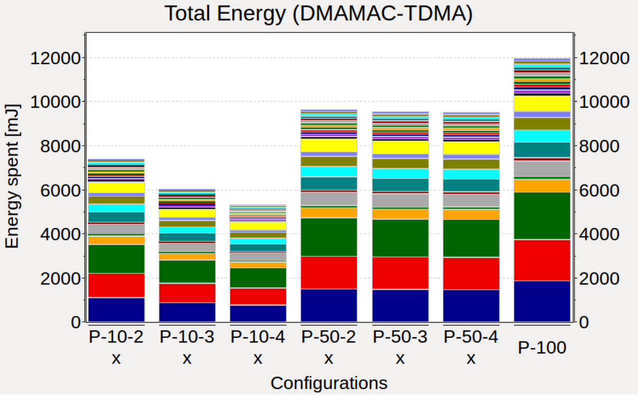


Figure 10: Energy consumption DMAMAC-TDMA

done by the sink.

4.2 Energy Consumption

The DMAMAC protocol is designed for a tree topology which means that the energy consumption is not uniform across all nodes as discussed in Sect. 3. Thus, we consider *total energy* as the metric to measure and compare total energy consumption of the DMAMAC protocol and the GinMAC protocol. Total energy is the energy spent by all the nodes in the network (except the sink which is assumed to be wire powered) during the entire simulation. We compare all configurations, and variations of the DMAMAC protocol (Hybrid and TDMA) with GinMAC. Firstly, we present the total energy consumption across various configurations of the two variants DMAMAC-Hybrid and DMAMAC-TDMA. Graphs depicting the results are shown in Fig. 9 and Fig. 10. Note that, in these figures, the colors across each configuration represent different nodes. This is to highlight the difference in energy consumption among the nodes. Also, Table 4 lists the numbers for energy consumption for both the variants of the DMAMAC protocol. The comparison column (Relative) in the table represents energy spent using DMAMAC protocol (E_{DMAMAC}) in percentage of energy spent using GinMAC protocol (E_{GinMAC}). The total energy consumption in the network using the GinMAC protocol is $E_{GinMAC} = 11,950$mJ or 11.95J. The average value of the energy consumed over 100 runs is used for the comparison.

The graph in Fig. 11 shows the comparison for total energy consumption. The energy consumption of the network using the GinMAC protocol is represented as a black line in the graph. The energy consumption for the GinMAC protocol is obtained for a single configuration and a single run. Given the static nature of the GinMAC superframe (no random elements like alert message) running multiple simulations does not yield any difference. Also, the GinMAC superframe is fixed for a given network, and cannot be extended/modified as in the case of the DMAMAC transient superframe. Thus only one configuration is possible with GinMAC. We can observe that DMAMAC-TDMA is the most energy-efficient of all protocols compared across all configurations tested for the DMAMAC protocol. This is due to energy consumption on nodes that have to be awake for the entire alert slot duration in DMAMAC-Hybrid. Given the different configurations, it is possible to adapt the protocol design based on the application requirement. The length of the steady superframe can be varied to obtain higher energy efficiency or higher data rate (smaller steady superframe). Also, note that the results of 50% transient operation is primarily shown to get a wider perspective on results obtained. In general DMAMAC is aimed at serving applications where steady state is dominant ($\geq$ 90%).

	DMAMAC-Hybrid		DMAMAC-TDMA	
Config.	Avg.[J]	Relative [%]	Avg.[J]	Relative [%]
P-10-2x	8.90	74.48	7.39	61.84
P-10-3x	7.99	66.86	6.00	50.20
P-10-4x	7.54	63.10	5.32	44.52
P-50-2x	10.25	85.77	9.65	80.75
P-50-3x	10.11	84.60	9.53	79.75
P-50-4x	10.06	84.18	9.5	79.50

Table 4: Comparing total energy consumption

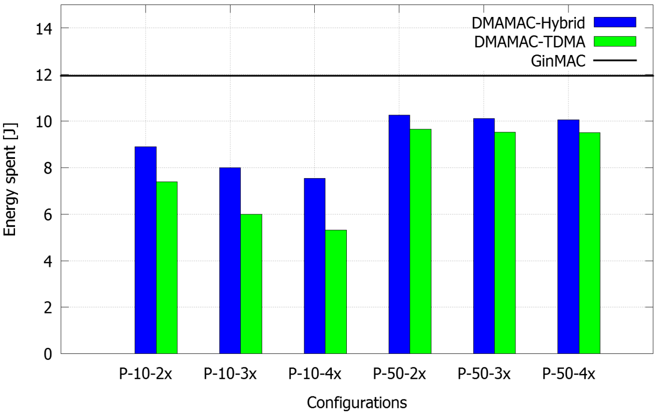


Figure 11: Comparing both variants of DMAMAC with GinMAC for total energy consumption

4.3 Network Lifetime

The network lifetime is a measure of the survivability of the network on a single battery charge. This can be viewed in several different ways, one of which is time to first node death. Using this approach, we have evaluated GinMAC and DMAMAC. The graphs in Fig. 12 and Fig. 13 show the time to first node death for the two variants of the DMAMAC protocol across different configurations. The x-axis gives the relative time since the evaluation is based on the initial battery capacity which may vary depending on the particular battery used on the nodes. Node 3 (see

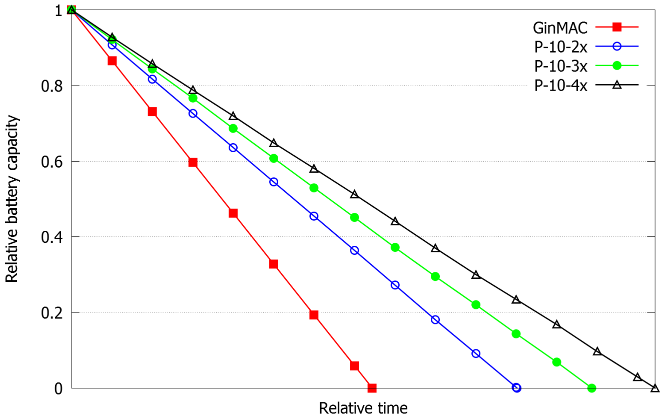


Figure 12: Network Life Time (DMAMAC-Hybrid)

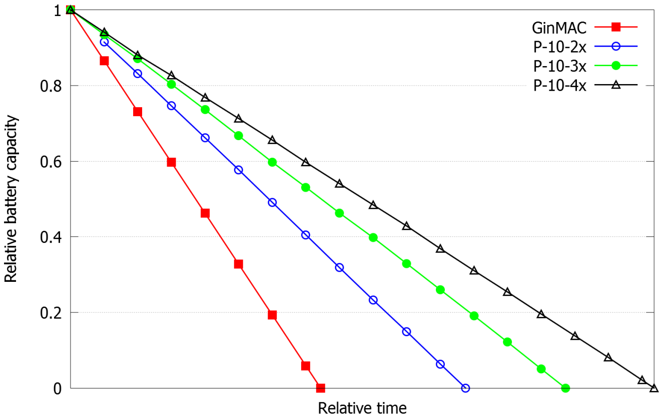


Figure 13: Network Life Time (DMAMAC-TDMA)

Fig. 5), is the node that dies first due to its position in the tree topology which causes it to have the highest number of ancestors in the network. The simulation was conducted with a fixed initial battery capacity, and run until node 3 had depleted all its battery. Given the focus on the 10% transient configuration, we have skipped 50% configurations in these graphs.

A comparison between the two variants of the DMAMAC protocol and GinMAC is shown in Fig. 14. It shows the relative time to death for node 3 across different configurations. The time to death for node 3 with GinMAC is represented with a black line on the graph. In Table 5, we list all simulation configurations and the energy consumption for node 3 obtained from the simulation. The average energy consumption of the node is presented in the table. Among all the explored configurations, the 4x configuration of the TDMA-variant of DMAMAC is the most energy efficient. Also, overall the TDMA-variant fares well in comparison with GinMAC and the Hybrid-variant. It is also important to note that the time from an alert being generated and the state-switch happening is the same in all these configurations as discussed in Sect. 2.

4.4 State-switch Failures

The switch of operational modes is an integral part of the DMAMAC protocol, and particularly the switch from steady to transient mode is critical. The aim of an ideal MAC protocol is to ensure that this switch happens whenever alert messages are generated or basically when the process moves to the transient state.

4.4.1 DMAMAC-Hybrid

In the DMAMAC-Hybrid variant, collision is inevitable with the use of CSMA in the alert slots. These collisions could result in state-switch failures when the alert packet is lost due to collision, and the sink is hence not notified of alerts. Alert messages exhibit random behavior, i.e., it is possible that several sensors detect the change of process states. Further, when two or more sensors send the data towards the sink, collision could occur if their parent node is the same, and both have comparably the same random delay. This results in the alert message being lost.

In Table 6, we present the results for state-switch failures for different configurations. Also, a graph representation of the switch-attempt failure in comparison to the collisions and the total number of switches is shown in Fig. 15. The aim of our DMAMAC protocol is to keep the state-switch failure within [0-1]%, mainly for the configurations of P-10 (2x,3x and 4x). From the results presented, considering

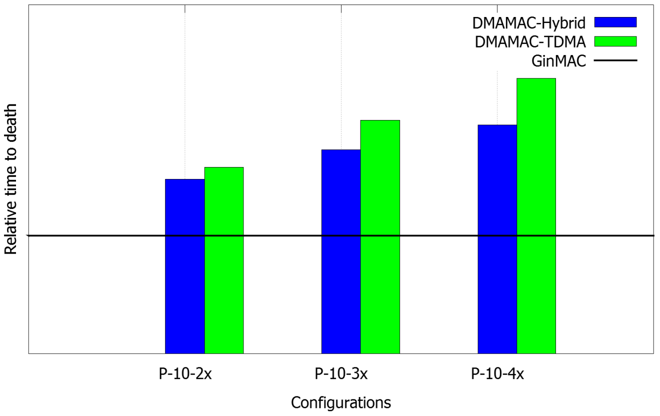


Figure 14: Time to death comparison between different protocols

Energy spent by Node 3 on GinMAC = 2.16 J				
	DMAMAC-Hybrid		DMAMAC-TDMA	
Config.	Avg. [J]	Relative [%]	Avg. [J]	Relative [%]
P-10-2x	1.40	64.94	1.30	60.30
P-10-3x	1.18	54.73	1.04	48.24
P-10-4x	1.06	49.17	0.91	42.21
P-50-2x	1.73	80.24	1.74	80.71
P-50-3x	1.70	78.85	1.72	79.78
P-50-4x	1.69	78.39	1.71	79.31

Table 5: Node 3 energy consumption

Config.	Failed Switches (avg)	Successful Switches (avg)	Relative [%]
P-10-2x	0.23	76.13	0.30
P-10-3x	0.15	76.17	0.12
P-10-4x	0.07	76.62	0.10
P-50-2x	6.73	199.14	3.38
P-50-3x	22.20	199.52	11.18
P-50-4x	21.77	196.17	11.10

Table 6: State-switch attempt failures

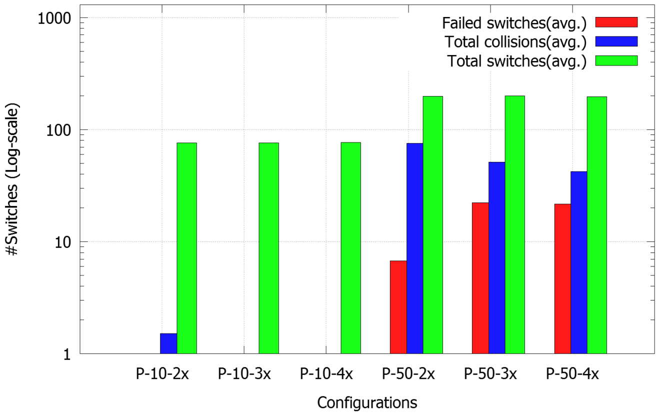


Figure 15: Switch failure for 25 nodes

the configurations P-10-2x, P-10-3x and P-10-4x, the state-switch failure is at most 0.23 % (i.e., within [0-1]%) or 2.3 failures in 1000 state-switches (on average). An alternative method to prevent state-switch failures could be proper identification of collisions, as collision of actual alert packets and not random noise, and considering them as alert.

Even if a state-switch attempt fails, the state-switch would eventually happen since nodes can store the information about a threshold being violated, and then notify the sink in the next alert. But this goes through the same process, and has the same probability of state-switch failure. Given the low probability of state-switch failure obtained, the expected state-switch delay is low. Assuming that the probability of failure is independent, the number of expected alert messages before switch can happen equals $\frac{1}{0.9977} = 1.0023$ (based on the 0.23% of 10-2x configuration). Hence we would expect that the state-switch happens in the immediate next alert at worst. The configurations with 50% transient state have a much higher failed number of switches, and are presented here for a broader view of the performance.

4.4.2 DMAMAC-TDMA

The TDMA-variant of the DMAMAC protocol eliminates the collision possibility. Thus in ideal conditions, switch failure does not exist. But for packet failure conditions in a real wireless channel, alert messages must be used along with re-transmission similar to data transmission part in the DMAMAC protocol. This introduces an extra energy expenditure into the existing energy consumption, but can be used for increasing reliability of switches.

4.5 Packet Transmission Delay

The end-to-end (sensor-to-actuator) transmission delay depends on the superframe structure, since it is of static nature. For the considered topology, shown in Fig. 5 consisting of 25 nodes with 1 re-transmission slot for each sensor/actuator data slot, the maximum end-to-end transmission delay is 1200ms (1.2s). This is the time interval between the first sensor sending the data until the last actuator receives the data. We have a lower delay compared to the GinMAC superframe where the actuator slots are placed towards the end of the superframe [15]. But this is a design choice and can be appropriately modified in GinMAC as well. Alternatively, the delay can be reduced or increased by changing the number of re-transmission slots used in transient and steady mode.

4.6 Maintaining Reliability on lossy links

In the DMAMAC protocol, the re-transmission slots in general are based on the re-transmission slots in GinMAC. But given that we have a dual-mode protocol, we can vary the re-transmission slots in transient and steady mode such that they are different. This could in general be done to further increase energy efficiency, i.e. relatively lower number of re-transmission slots in steady than in transient mode. Alternatively, a larger number of re-transmission slots in the steady state could be considered to make sure the data reaches the actuator with a higher probability, but at a higher energy cost. In the current study, we used the same number of re-transmission slots for both GinMAC and DMAMAC protocol, and the same number for both operational modes of the DMAMAC protocol. The re-transmission slots are used in the simulation to give an idea of possible delay caused due to the usage of re-transmission slots. There is no difference between the performance of the two protocols in terms of reliability. Concerning the sink's single notification slot used, the loss of a notification packet can cause a difference in operational modes between different nodes and sink. In general this can be prevented by using dedicated repeaters to assist sink signals in reaching the nodes correctly.

4.7 Scalability

Scalability of MAC protocols is a general challenge in WSN and WSAN. For the DMAMAC protocol, we suggest a maximum of 25 nodes similar to GinMAC for a single sink to keep the delay low for process monitoring and control applications. In case a larger number of nodes are required, a backbone connecting multiple sub-networks of 25-nodes each managed by a separate sink is suggested. The backbone may use powerful high-rate data transfer (wired or wireless). This can address the scalability issue to some extent. In principle, larger networks can be managed in case the delay requirements permit it. More the energy spent in transient state, the better is the relative energy efficiency of the DMAMAC protocol (both variants) than that of GinMAC. This indicates that the increase in packet size and/or the number of nodes in the network, could result in large energy savings with the use of the DMAMAC protocol. From this perspective, the DMAMAC protocol can be considered as scalable.

4.8 Hybrid and TDMA

On an ending note of the performance evaluation, we can observe that DMAMAC-TDMA has better energy efficiency among the two variants proposed. We preserve the representation of the DMAMAC-Hybrid to encourage future investigations to use its hybrid nature effectively. In particular for larger networks where the topology is horizontal and the alerts are less frequent, the hybrid variant may have better energy efficiency than TDMA. But it is still prone to collisions, and switch delay issues resulting from collision.

5. CONCLUSIONS AND FUTURE WORK

The DMAMAC protocol is aimed at satisfying real-time requirements of process control systems while preserving energy to prolong network lifetime. The DMAMAC protocol design has been improved in comparison to the previous version based on initial simulation results and analysis. We simulated both variants of the DMAMAC protocol using the OMNeT++ platform in conjunction with MiXiM libraries for wireless networks to evaluate the performance of the protocol. The DMAMAC protocol shows considerable reduction in energy consumptions compared to the GinMAC protocol for process control systems with dominating steady state. In particular, the DMAMAC-TDMA exhibits the lowest energy consumption among all. We mainly compared the DMAMAC-Hybrid,DMAMAC-TDMA, and the GinMAC protocols based on metrics of total energy and network lifetime. For network lifetime, we considered time to first node death. Given the random nature of alert messages in DMAMAC-Hybrid, we have also evaluated this protocol variant for state-switch failures to give a better idea of directions for further study. The state-switch failures are $\leq$ 0.23% for configurations of 10% transient superframes which is tolerable. The study of varying re-transmission slots in steady and transient superframes could add on to the simulation results.

The DMAMAC protocol is an application specific protocol proposed for process monitoring and control applications. The idea can be applied across domains for similar monitoring and control applications that have event-based traffic conditions. Two possible application domains are monitoring and control in healthcare and home automation. Given the nature of applications in these domains and the number of nodes used, the requirements match the design constraints of the DMAMAC protocol. Given the requirements and challenges existing in Wireless Body Area Networks (WBAN) [9], DMAMAC protocol could be used to address some of the challenges along with prime focus on energy efficiency. In WBAN, mainly two types of data are handled: periodic monitoring data and emergency event based data [2]. The dual mode operation allows for facilitating high data rate communication for priority traffic during emergencies and otherwise using low data rate to facilitate energy efficiency prolonging the lifetime and reducing the cost. The applications in the healthcare domain could particularly benefit from the reliability and energy efficiency provided by the TDMA-variant. One such application is the sensor-actuator implementation to maintain glucose level in a diabetes patient, which consists of glucose sensors and insulin actuators. Adaptation for non-critical parts of home automation would be easier considering that the application is far less critical, and thus switch delay requirements would not be as stringent. Such applications could benefit from the use of either variants of the DMAMAC protocol. This could include temperature management systems. But critical or emergency systems including fire-safety and theft-safety in home automation could benefit from the TDMA-variant. Thus in general, the dual-mode operation and other features of DMAMAC protocol can be applicable across domains in internet of things [12], but would require adaptation based on requirements.

Near future work in the development of DMAMAC is to create an implementation of DMAMAC and perform deployment testing. This would further validate the applicability of the protocol in real process control systems and also, provide insights on the differences between the results obtained from simulation and implementation.

6. REFERENCES

[1] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, 2004.

[2] H. Alemdar and C. Ersoy. Wireless sensor networks for healthcare: A survey. *Computer Networks*, 54(15):2688 – 2710, 2010.

[3] A. Bachir, M. Dohler, T. Watteyne, and K. Leung. MAC essentials for wireless sensor networks. *IEEE Comm. Surveys & Tutorials*, 12(2):222–248, 2010.

[4] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu. A survey of recent results in networked control systems. *IEEE Proceedings*, 95(1):138–162, 2007.

[5] T. Instruments. Chipcon CC2420 datasheet. 2007.

[6] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K. Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in OMNeT++ the MiXiM vision. In *Proc. of SIMUTOOLS*, pages 1–8, 2008.

[7] T. O Donovan, J. Brown, U. Roedig, C. J. Sreenan, J. do O, A. Dunkels, A. Klein, J. Silva, V. Vassiliou, and L. Wolf. GINSENG: Performance control in wireless sensor networks. In *Proc. of SECON*, 2010.

[8] A. Onat, T. Naskali, E. Parlakay, and O. Mutluer. Control over imperfect networks: Model-based predictive networked control systems. *IEEE Transactions on Industrial Electronics*, 2011.

[9] M. Patel and J. Wang. Applications, challenges, and prospective in emerging body area networking technologies. *IEEE Wireless Communications*, 17(1):80–88, February 2010.

[10] K. Pister and L. Doherty. Tsmp: Time synchronized mesh protocol. *IASTED Distributed Sensor Networks*, pages 391–398, 2008.

[11] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu. Z-mac: A hybrid mac for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 16(3):511–524, June 2008.

[12] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. McCann, and K. Leung. A survey on the ietf protocol suite for the internet of things: standards, challenges, and opportunities. *IEEE Wireless Communications*, 20(6):91–98, December 2013.

[13] A. A. K. Somappa, K. Øvsthus, and L. M. Kristensen. Towards a dual-mode adaptive MAC protocol (DMA-MAC) for feedback-based networked control systems. *Procedia Computer Science*, 34(0):505–510, 2014. ComSense 2014.

[14] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, and M. Nixon. Wirelesshart: Applying wireless technology in real-time industrial process control. In *IEEE Real-Time and Embedded Technology and Applications Symposium, 2008*, pages 377–386, April 2008.

[15] P. Suriyachai, J. Brown, and U. Roedig. Time-critical data delivery in wireless sensor networks. In *Proc. of DCOSS*, pages 216–229, 2010.

[16] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proc. of SIMUTOOLS*, pages 1–10, 2008.

Chapter 9

Model-based Specification and Validation of the Dual-Mode Adaptive MAC Protocol

Model-based Specification and Validation of the Dual-Mode Adaptive MAC Protocol

Abstract:

Wireless Sensor and Actuator Networks (WSANs) rely on MAC protocols to coordinate access to the wireless medium access and for managing the radio unit on each device. The Dual-Mode Adaptive MAC (DMAMAC) protocol is a recently proposed protocol designed to reduce the energy consumption of the radio communication in WSANs. The DMAMAC protocol targets the industrial WSANs used for real-time process control. At its core, DMAMAC exploits the distinction between transient and steady of the controlled plant process to dynamically adapt the MAC superframe structure and thereby conserve energy. The switch between steady and transient mode of operation is a safety-critical part of the protocol. The contribution of this paper is to develop a rigorous specification of the DMAMAC protocol using timed automata and the supporting Uppaal software tool. The Uppaal tool is also used to verify key functional and real-time properties of the protocol. Finally, we have used execution sequences from the formal specification model to validate an OMNET simulation model used for performance evaluation of DMAMAC, and to validate a nesC implementation of the protocol on the TinyOS platform.

Keywords: Verification, Validation, Simulation, Deployment, Industrial Wireless Sensor Network, MAC protocol, Uppaal

Biographical notes:

This paper is a revised and expanded version of a paper entitled "Model-Based Verification of the DMAMAC Protocol for Real-time Process Control" presented at the 9th Int. Workshop on Verification and Evaluation of Computer and Communication Systems, Bucharest, September 2015.

ï»¿

1 Introduction

A Wireless Sensor Actuator Network (WSAN) [1] consists of sensors and actuators that use radio communication to send, relay, and receive information. WSANs are used in a wide range of domains including process- and factory automation, smart home automation, and health-care. Feedback-based control loops that use wired or wireless solutions are collectively known as Networked Control Systems (NCS) [2]. NCS mainly use wired communication systems, but are increasingly adopting wireless communication. The salient feature of wireless solutions is the reduction in cost and size compared to the use of wired networks. The use of wireless communication, however, also has challenges and it has not yet become the de-facto replacement for wired solutions. The challenges of wireless solutions include low-bandwidth, energy efficiency, signal interference, and packet-loss. Energy efficiency in particular is an important concern when devices are battery powered.

Wireless solutions are made up of a collection of protocols that cater for different services. Medium Access Control (MAC) is one of the functions that are critical to the proper operation of the entire WSAN. MAC protocols govern the communication and control the use of the radio module on each node in the network which is the dominant consumer of energy in wireless nodes. The Dual-Mode Adaptive MAC (DMAMAC) protocol [3] is a recently proposed MAC protocol for process control applications. The protocol is aimed to support an energy efficient wireless solution. The DMAMAC protocol was proposed for NCSs with real-time and energy efficiency requirements. In particular, it targets process control applications that differentiate between two states of operation: steady and transient. Fig. 1 shows a typical process control with two states. The transient state corresponds to the process state with large and frequent change in measurements of physical quantities, resulting in a high data transfer rate. The steady state refers to the process state with measurements contained within a controlled range of values, and thus requiring less data transfer. An example is process control for chemical reactors where the measure physical quantities typically include temperature and pressure measured by sensors. These physical quantities can be controlled by means of actuators that controls the inflow of chemicals and coolant to the chemical reactor. Initially, the DMAMAC protocol was designed using a combination of Carrier Sense Multiple Access (CSMA) and Time Division Multiple Access (TDMA). Based on simulation studies [4] a full TDMA version (DMAMAC-TDMA) has been designed. The advantage of the full TDMA version is that it is more predictable and robust towards packet failure which is highly relevant for safety-critical applications. The hybrid CSMA-TDMA variant on the other hand supports a smaller state-switch delay and is in some scenarios more energy efficient.

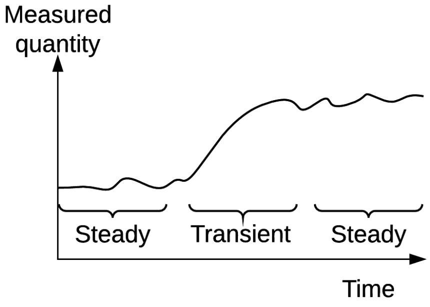


Figure 1: Process control states

It follows from the above that the DMAMAC protocol is to be used for safety-critical applications. In particular, the state-switch is a safety critical feature of the DMAMAC protocol that can benefit from formal verification to ensure proper functioning. This has motivated us to apply formal modelling and model-checking which constitute a well-known technique for verification of protocol designs. Model-checking allows for exhaustive verification and has been widely used on related protocols (see, e.g., [5, 6, 7]). Verification in the early design phase can be used to ensure the behavioural correctness of protocols. Model-checking assists in discovering design faults by exhaustively traversing all possible execution traces of a given model. Furthermore, model-checking tools can provide traces to failure states, thus assisting in resolving any discovered design issues. Uppaal [8] is a modelling and verification tool-suite that supports model checking of real-time systems. In addition to model-checking and verification, Uppaal also supports simulation which can be used to provide useful insights into the operation of a protocol.

In this article, we apply the Uppaal tool to analyse qualitative features of the DMAMAC protocol. We present a formal specification of the DMAMAC protocol in the form of a network of timed automata and verify safety properties related to the absence of faulty states. We verify real-time properties including switch delay and maximum data delay. The timed modelling of the DMAMAC protocol is based on a Finite State Machine (FSM) representation of the sensors, actuators, and the sink node in the WSAN network configuration under consideration. We consider both the hybrid and the TDMA-version of the DMAMAC protocol. Furthermore, we use the constructed Uppaal models to validate a OMNeT++-MiXiM [9] simulation model of DMAMAC and our [10] prototype implementation of DMAMAC. The OMNeT++-MiXiM simulation model has been used to study the performance of the DMAMAC protocol [4]. The validation of the simulation model and the implementation is done via the comparison of sequence diagrams extracted from executions of the Uppaal model, the OMNET++ model, and the TinyOS implementation.

Uppaal has been widely used to model and verify communication protocols (see, e.g., [5, 7, 6]). The Lightweight Medium Access Control (LMAC) [6] protocol is the closest MAC protocol modelling related to the work presented in this article. The LMAC and the DMAMAC protocols are two distinct protocols with distinct goals, and differ significantly in their base features. The LMAC protocol is a self-organising protocol with nodes selecting their own slots i.e., time duration allocated for data transfer. The focus in the LMAC protocol verification is on efficient slot selection and collision detection. In the DMAMAC protocol, the slot scheduling is done statically and offline prior to deployment. The focus of the DMAMAC protocol is to provide an energy efficient solution along with efficient switching between the two operational modes. It requires a different model to represent the features of the DMAMAC protocol than the one used for the LMAC protocol. In [7], the authors have focused mainly on modelling the Chipcon CC2420 transceiver. This work is related in terms of their use of a packet collision model and how collisions are observed. We use a collision model similar to [7, 6]. With the extension of Statistical Model-Checking (SMC) features, Uppaal can also be used to assess performance related queries as shown in the case study [8] of the Lightweight Medium Access Control (LMAC) protocol.

The rest of the article is organised as follows. In Sect. 2 we briefly introduce the DMAMAC protocol. For extensive details, we refer to [3]. Section 3 describes in detail the constructed Uppaal model of the DMAMAC protocol. As part of this, we briefly introduce the constructs of timed automata as supported by Uppaal, and perform some initial validation of the protocol model. In Sect. 4 we complete the validation of the constructed model. The verification of the protocol for different deployment configurations is discussed in Sect. 5. In Sect. 6 we use execution sequences from the Uppaal model to validate an OMNET++ simulation model of DMAMAC and an implementation of DMAMAC. Finally, in Sect. 7 we sum up the conclusions and discuss future work. The reader is assumed to be familiar with the basic concepts and operation of MAC protocols, including superframes and slots, and the principles of TDMA and CSMA. The current article is an extension of the work presented in [11].

ï»¿

2 The DMAMAC Protocol

The DMAMAC protocol [3] has two operational modes catering for the two states of process control applications: transient mode and steady mode. The hybrid variant of the protocol is based on Time-Division Multiple Access (TDMA) for data communication and a combination of Carrier Sense Multiple Access (CSMA) and TDMA for alert message communication. The TDMA variant uses TDMA slots for both data communication for alert message communication. The basic functioning of the DMAMAC protocol is based on the GinMAC protocol [12] proposed for industrial monitoring and control. The network topology of the DMAMAC protocol consists of sensor nodes, actuator nodes, and a sink. The sensor nodes are wireless nodes with sensing capability which sense a given area and update the sink by sending the sensed data. The actuator nodes are wireless nodes equipped with actuators, which act on the data performing a physical operation. It is also possible to have wireless nodes with both sensors and actuators. The sink is a computationally powerful (relative to the nodes) wire-powered node which collects the sensed data, performs data processing on it, and then sends the results and commands to the actuators.

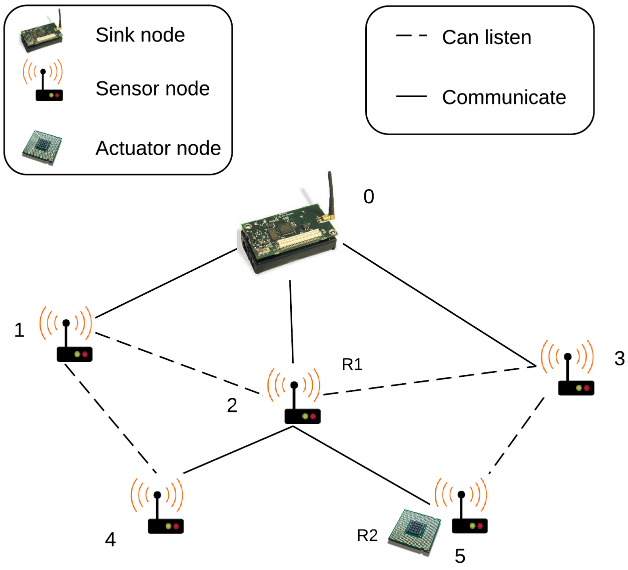


Figure 2: The network topology for DMAMAC protocol

Similar to the GinMAC protocol, the network deployment for the DMAMAC protocol is based on a tree topology as shown in Fig. 2. The solid lines between nodes represent data communication. The dashed lines represent nodes that are within transmission range of each other, but which have no direct data communication in the network. Each level in the tree topology is ranked (marked with “R#", where # is either 1 or 2), with the sink having the lowest rank number and the most distant nodes from the sink have the highest rank number. This ranking is exploited in the alert message sending procedure.

The design of the DMAMAC protocol has been based on the following assumptions:

- The nodes are assumed to be time synchronised via an existing time synchronisation protocol.
- The sink is assumed to be powerful, and it can reach all nodes in one hop.
- A pre-configured static network topology with no mobility is assumed.
- A single slot accommodates both a data packet and a corresponding acknowledgement.

Time synchronisation is a vital part in TDMA-based WSANs and the first assumption makes DMAMAC independent of the particular protocol used for time synchronisation.

Below we briefly explain the working of the two operational modes and the respective superframes they use. Both of the DMAMAC variants rely on the same transient mode superframe structure, but differ in the steady mode superframe structure.

2.1 Transient Mode

The transient mode is designed to imitate the transient state operation in process control. In the transient state, the controlled process changes rapidly generating data at a faster rate relative to the steady mode. During the transient mode operation, the DMAMAC protocol uses the transient superframe shown in Fig. 3. The superframe includes a data part for data transfer from the sensors to the sink, followed by a data part with data being sent from the sink to the actuators, and then a sleep part. The data part also includes a notification message slot from the sink to all nodes, and a sink processing slot. A typical transient mode operation cycle consists of:

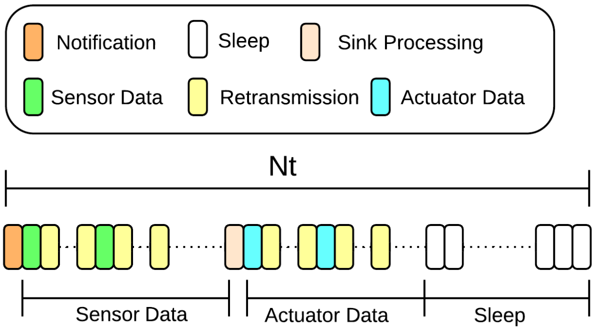


Figure 3: The transient superframe of the DMAMAC protocol

1. A notification message is being sent from the sink to all the nodes.
2. The data part is executed with data transmission from the sensors to the sink and then to actuators.
3. The sleep part is executed where all sensors and actuators enter sleep mode in order to improve energy efficiency.

The notification message in item 1 includes control data like state-switch message and time-synchronisation. This sleep part in item 3 represents the situation where all nodes are in

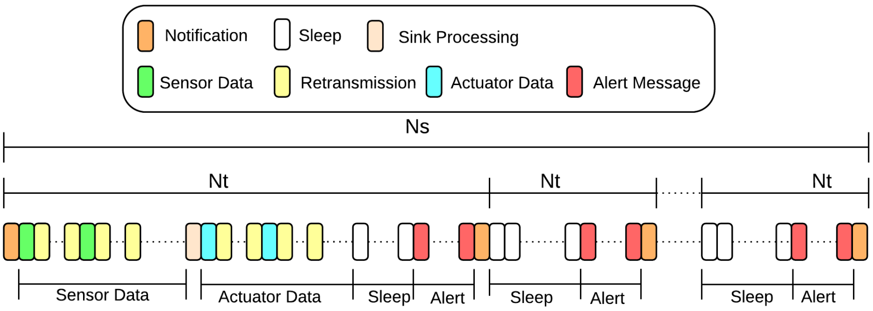


Figure 4: The steady superframe of the DMAMAC protocol

sleep mode. Individually, the nodes are in sleep mode when they are not performing other tasks.

2.2 Steady Mode

The steady mode operation is designed to operate during the steady state of the controlled process. The steady superframe used in the steady mode operation is shown in Fig. 4. In addition to the parts that also exist in the transient superframe, the steady superframe contains an alert part. The alert part is used to ensure that the state-switch from steady to transient mode occurs whenever a sensor detects a threshold interval breach. This threshold is set by the sink when the switch from transient to steady is made. It should be noted that in comparison to [3], a slightly modified steady mode superframe is used in that the notification slots placed at the end of each transient (Nt) part. This is done to facilitate immediate application of alert, and making a state-switch. In the alert part, one slot is allocated to each node with the same rank. All the nodes in the same rank have the possibility to send an alert message in this slot. A typical steady mode operation cycle is as follows:

1. A notification message is followed by the data and the sleep part. This is similar to the working in transient mode operation.
2. Sensor nodes that have alert messages to be communicated use appropriate slots provided for each rank to notify parents about the alert.

The alert messages in item 2 is relayed towards the sink which then makes the switch to the transient state. In an absence of alert, sensor nodes still wake up on their alert receive slot and then enter sleep mode until the next notification slot. In the alert part, the notification slot is placed at the end. This is done to ensure a quick transition between the two states. All regular nodes wake up in this slot, and receive a notification message from the sink. Alert notification to change superframes is also sent here.

2.3 Dynamic Change of Superframes

The distinct feature of DMAMAC compared to other MAC protocols for WSANs is the ability to dynamically switch the MAC superframe being used depending on the state of the controlled process. There are two switches possible: transient to steady and steady to

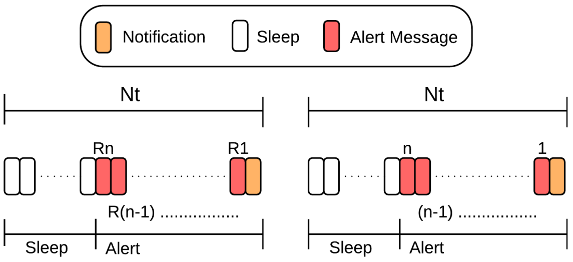


Figure 5: The alert part for DMAMAC-Hybrid (left) and DMAMAC-TDMA (right)

transient. The latter is a safety-critical switch since the data rate in transient is higher and it is important to accommodate the higher data rate in the transient state in order to control the process. The switch from transient to steady is decided by the sink, which determines if the process is in steady state based on received sensor readings. When the sink decides to make the switch, it informs all the nodes in the network to change their mode of operation. The message is sent via a notification message from the sink. When the sink node switches from transient to steady, it defines a threshold interval within which the sensor readings should lie, and informs the sensors about this threshold interval. During the entire steady mode operation, the sensors constantly monitor for a threshold breach. When there is a breach, the sensor node waits until its alert slot, then notifies its parent, which in turn forwards the alert towards the sink. The sink then informs the nodes in the network to switch to transient in its immediate next notification message.

2.3.1 DMAMAC-Hybrid versus DMAMAC-TDMA

The alert message is the message created by the sensor nodes to notify the sink that a state-switch is required. After the simulation studies conducted in [4] of the hybrid DMAMAC protocol, a pure TDMA-based variant of DMAMAC-TDMA was also developed that has more predictable behaviour in terms of ensuring that the state-switch takes places. The predictability comes at the expense of being less energy efficient than the hybrid variant. The difference between the hybrid variant (DMAMAC-Hybrid) and the pure TDMA-based variant (DMAMAC-TDMA) is the way alert messages are handled.

In DMAMAC-Hybrid, the sensor nodes choose a random delay in the slot before transmitting the alert message. At the completion of the time duration of the random delay, the nodes sense the channel to prevent collision. If a node during channel assessment detects another node sending an alert message, then it just drops its alert message. Collisions are still possible, e.g., when two nodes choose the same random delay or when two senders cannot listen to each other but the receiver can listen to both. The nodes check for a change of operational mode following the sending of the alert. If no change occurs (because of collision) the nodes save the alert and send the alert again in the next alert slot. The structure of the alert part for DMAMAC-Hybrid is shown in Fig. 5 (left) for the network topology presented earlier in Fig. 2. The alerts slots are divided based on the levels (R#) of the network topology which means that all nodes in a given rank have the same alert slot.

DMAMAC-TDMA uses time divided slots also for alert messages. Thus, all nodes have their own alert slots removing the possibility of interference or collision within the same network elements. The rest of the superframe structure is the same as for DMAMAC-Hybrid. The alert message slots for the DMAMAC-TDMA variant is shown in Fig. 5 (right) where each node $1 \ldots n$ now has its own alert slot. This structure reduces the non-determinism in the protocol and provides more predictable performance. It should be noted that there could still be interference or collision due to parallel networks operating in the vicinity and it is still possible to loose packets in transmission.

3 The DMAMAC Uppaal Model

Uppaal [8] is an integrated tool-set based on timed automata for model-checking of real-time systems and supporting modelling, simulation, validation, and verification [13]. An abstract representation of a real-time system in the form of a model is structured as a network of timed automata. The query language of Uppaal allows for verification of safety, reachability, liveness, and time-bounded properties. In Uppaal, models are constructed as a network of templates based on timed automata. Templates are used to represent independent entities (e.g. a sensor node). Uppaal consists of two simulators: a symbolic and a concrete simulator. The symbolic simulator is used to inspect the execution of the model step by step. For certain queries, Uppaal outputs traces which can be viewed in the symbolic simulator. This is useful for pin-pointing error locations and sequences of events that lead to errors/faults. The symbolic simulator also shows all the templates in the model, and message sequence charts (MSCs) can be used to visualise communication between different processes. The symbolic simulator also allows interactive step-wise simulation of the model. Along with features similar to the symbolic simulator, the concrete simulator has the added advantages of firing transitions at a specified time.

We have developed Uppaal models for both variations of DMAMAC, i.e., DMAMAC-TDMA and DMAMAC-Hybrid. The hybrid variant has three source of non-determinism: triggering of alerts from sensor nodes, alert sending delay, and collisions. In the Uppaal model of DMAMAC-TDMA, the only source of non-determinism is the triggering of an alert in a node. Collision are not possible as per TDMA scheduling. The Uppaal models for these two variants only differ in the alert handling part. Thus, we only discuss the variation introduced in the alert part for DMAMAC-TDMA.

3.1 Modelling Decisions and Assumptions

We use a non-deterministic timed automata model to verify the properties of the DMAMAC protocol. The constructed model has several sources of non-determinism including the delay for sending alert messages in nodes, and the decision made by the sink to change from the transient mode to the steady mode. Given the design of alert messages, collisions are possible when sending alert messages. We use a simplified collision model, detailed later in this section. The sink and sensor/actuator nodes have separate timed automata models. Local clocks are used for each automaton. The main aim of the verification of the DMAMAC protocol model is to check that the two modes of operations are working correctly given the presence of non-deterministic choices (like collision) during execution and the delays that may occur.

The main assumptions and design decisions made during the construction of the Uppaal model for the DMAMAC protocol (both variants) are as follows:

- Packets are abstractly modelled without payload. The messages or packets exchange mechanism is represented by channel synchronisation in the Uppaal model.
- A global clock is used for a common network time reflecting the assumption on time synchronisation between the nodes. This can be considered as a way of abstractly implementing the time synchronisation between nodes assumed by the DMAMAC protocol.

For the hybrid variant of DMAMAC we additionally made the following modelling decisions:

- An exact model of CSMA results in a rather complex model. Instead, we use a representative CSMA procedure, which imitates the service and effects of actual CSMA on the working of the protocol. The effects include skipping packet transmission on detection of ongoing transmission and also collision. This makes our model and verification independent of the particular CSMA procedure that may be employed.
- The collision caused due to the use of CSMA has effects on the state-switch procedure. A simple collision model is used, where we record collision when two or more nodes send packets at the same time. Collision results in failure of the packets, thus affecting the state-switch procedure.

A channel synchronisation variable *choice* is used to force enabled transmissions. This is a modelling artefact and is not part of the protocol as such. In Uppaal, execution of models can remain in a location indefinitely even after outgoing edges are enabled. To force the model execution to continue via enabled outgoing edges, an urgent channel synchronisation is required.

3.2 Sink Model

The sink model is shown in Fig. 6. We have used colours in the automaton locations to differentiate between states. Both the sink and the node automata begin in an initial location **Start**. The sink initiates the startup procedure of the network using a broadcast synchronisation channel *startup* on the edge towards the **StateController** location. The function initialize() is used to set proper values to local and global variables. The sink reaches the **StateController** location upon having executed the startup procedure of the network. The node automata synchronise with the channel variable *startup*, and reach the **StateController** location.

The **StateController** location represents an event handler for handling transition between different states in the state-machine. The sink model uses a local clock variable x, which is active in all states indicated by $x' == 1$. The expression $x' == 0$ can be used to pause the clock counter. This is used as an invariant on all states to represent continuously running time. It also includes an invariant $x \leq currentMaxSlots * 10$ to prevent it from being in the state beyond the maximum timeframe of the active superframe (transient or steady). A typical slot time in WSANs is 10 milliseconds, and we therefore use the unit *ms* for time in our model. Given the time unit of *ms*, the variable *currentMaxSlots* is multiplied by 10 to obtain the slot-time. The use of the **StateController** location is also similar to the *self-message* handler in the commonly used OMNeT++ [14] framework for MAC protocols. The purpose of this particular handler is to check the self-message that it receives, and to act on the message by choosing an appropriate next state. Also, it determines the next state which is then sent as a self-message. The automaton changes between the different locations (states) in the model based on the local variable *currentSlot*, and the local clock variable x.

The **Notification** location is reached when the sink is due to send a notification message. The notification message is sent by the sink, and received by other nodes in the network. This is represented by the broadcast channel *regNotify[change]*, where *change* carries a message of the status of the Boolean variable *changeSuperframe*. The *changeSuperframe* variable is true when the sink needs to indicate to the nodes a change in superframes to switch the mode of operation. For both the steady to transient switch and the transient to steady switch, the sink uses *changeSuperframe*. An MSC generated from Uppaal for validating that notifications are correctly being sent and received in the model is shown in Fig. 7.

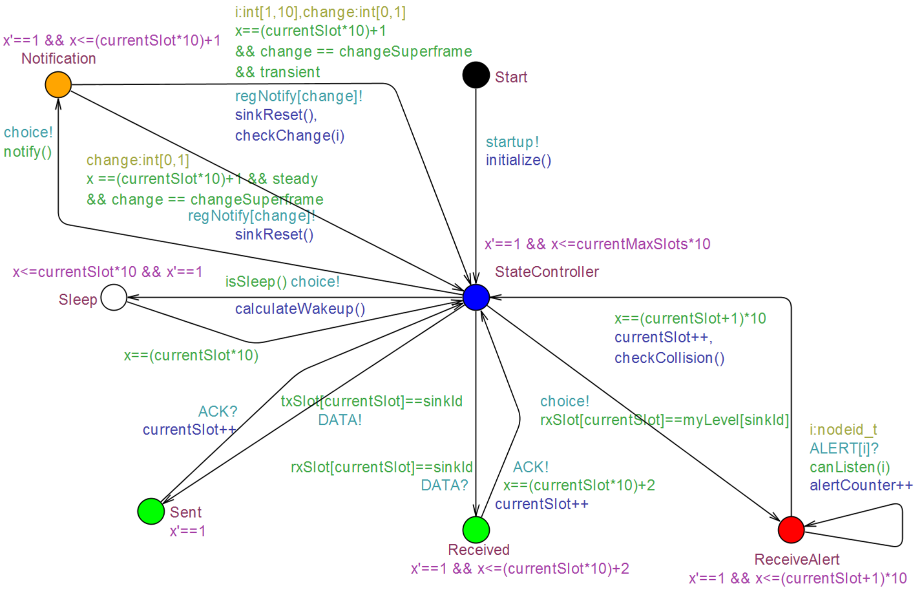


Figure 6: Uppaal timed automata model of the sink node

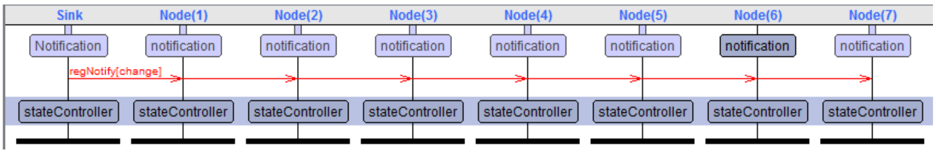


Figure 7: Message sequence chart for sending and receiving a notification

The switch from transient to steady is decided by the sink. There are two separate notification edges for transient mode and steady mode. In the transient mode, the sink decides if it has to switch to steady mode based on the random selection statement ($i : int[1, 10]$) and the obtained change value is sent over the channel. In the absence of real inputs, a random selection is used. The edge with the select statements $i : int[1, 10]$ and $change : int[0, 1]$ is used in transient mode. The second select statement ($change : int[0, 1]$) is a modelling artefact used to be able the sending of the value of *changeSuperframe* over the channel via the synchronisation variable *regNotify[change]*. The guard $change == changeSuperframe$ makes sure that the select statement selects the same value as the *changeSuperframe* variable.

In the steady mode, a switch is based on alert from nodes. The notification for the steady mode is done via the edge with only one select statement ($change : int[0, 1]$). As a symbolic representation, we have used a guard $x == (currentSlot * 10) + 1$ on these edges, and an invariant $x \leq (currentSlot * 10) + 1$ on location **Notification** to indicate a delay of 1 ms for message transmission. Both these edges use a function sinkReset(), which resets the sink variables at the beginning of a new superframe, and implements the changing of superframes.

The **Sleep** location is reached when the sink or nodes do not have any active operations to be conducted in the current slot. The edge to **Sleep** is guarded by a Boolean function isSleep() which checks if the current slot is a sleep slot. We use the urgent broadcast channel *choice* to force this transition whenever isSleep() evaluates to true. In the absence of this channel variable, the model can continue to be in the location **StateController** forever even when isSleep() is true. The location **Sleep** has an invariant $x <= currentSlot * 10$ which indicates that during execution, the control can be in the location as long as the time does not exceed the value *currentSlot*, which holds the value of the slot at which the sink should wake up for its next event. This is set by the function calculateWakeup() when **Sleep** is reached.

The location **Sent** is reached when the sink sends data in its data slot. The **Received** location is reached when the sink receives any sensor data, and then sends an ACK via channel synchronisation. Location **Received** has the invariant $x \leq (currentSlot * 10) + 2$ which is used to add a delay of 2 ms as a representation for the time required for data communication. A follow up guard on the ACK sending edge $x == (currentSlot * 10) + 2$ makes sure that the delay is applied. Upon sending or receiving of ACK synchronisation, the local variable *currentSlot* is incremented.

Lastly, the location **ReceiveAlert** is reached when it is the sink's turn to receive an alert. This is determined by the alert levels defined in the *myLevel* array variable represented by the guard $rxSlot[currentSlot] == myLevel[sinkId]$. The sink stays in the location for an entire slot duration (10 ms), and waits for any alerts from nodes it can listen to. The canListen(i) function is used as a guard to make sure that the sink listens to alerts from only those nodes that are in its listening range (same function used for nodes). The variable i given as input to the function is the result of a select statement $i : nodeid\_t$, which allows the node to listen to any node that is transmitting. The guard makes sure that the sink can listen to that particular node. At the completion of the alert slot, the sink checks if any collision has occurred via the function checkCollision(), and gives back the control to the **StateController**.

3.3 Sensor and Actuator Node Model

The sensor/actuator node model is shown in Fig. 8. The node model is similar to the sink model except for the notification handling procedure. Also, the node template consists of an extra location for sending alert messages. The notification part of the node model is simpler, since nodes only receive notifications. Location **Notification** is reached when the node is in its notification slot (receive) in either of the two types of superframes. The nodes then synchronize on the channel variable *regNotify[change]* from the sink, and reset the node variables using the function nodeReset() based on the value of the variable *change*.

The node model works similar to the sink model for sleep, sent (data), received (data), and alert receive. This means that in locations **Sleep**, **Sent**, **Received**, and **ReceiveAlert** the node and the sink model have the same modelling elements. An example of an MSC for validating the behaviour of the model for data transmission between nodes is shown in Fig. 9, and an MSC for data transmission towards sink is shown in Fig. 10.

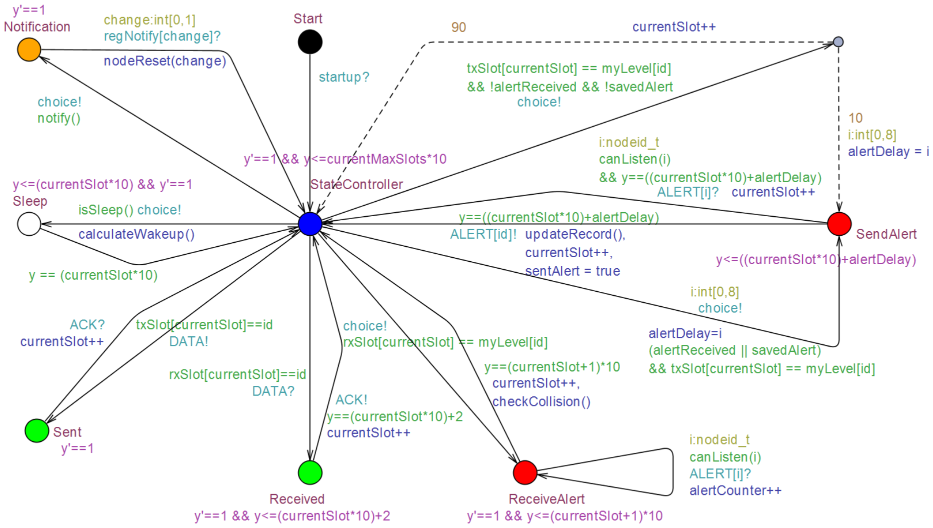


Figure 8: Uppaal timed automata model of a regular sensor/actuator node

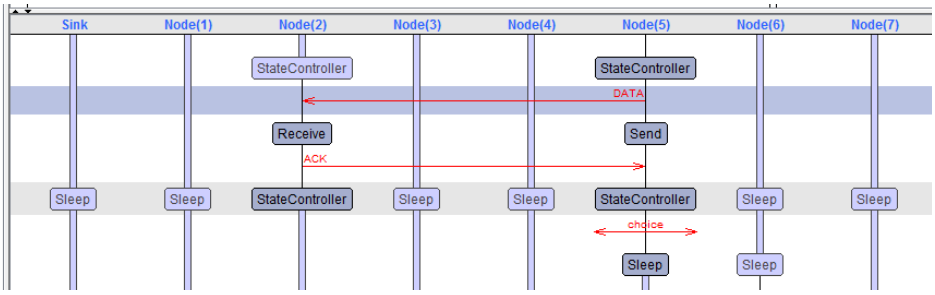


Figure 9: Message sequence chart for data transfer between nodes

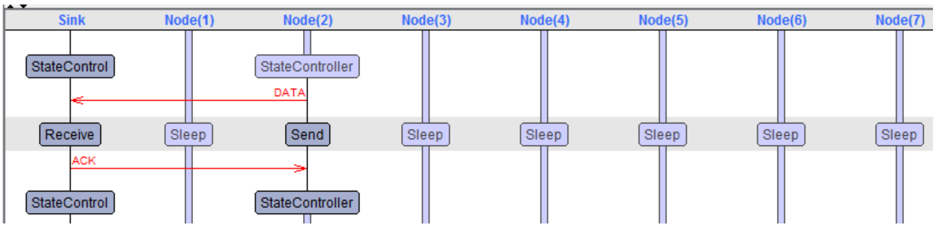


Figure 10: Message sequence chart for data transfer towards sink

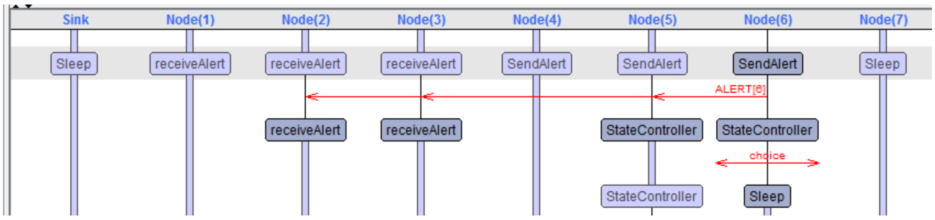


Figure 11: Message sequence chart for alert forwarding

The location **SendAlert** which handles the crucial part of alert message sending is required by the protocol for the switch of operational mode from steady to transient. Based on the protocol specification, a node can send an alert message when the sensed data crosses the threshold interval. This threshold interval is set by the sink depending on the particular process being controlled. We imitate this event using a probability weight-based selection for sending alert messages as reflected in the edge towards **SendAlert** (shown with dashed lines). The edge with probability weight 90 represents no alert to be sent. The one with probability weight 10 represents the choice to send an alert. The guards on the edge make sure it is the alert slot of the node, and that the node does not already have an alert to be sent. When the node chooses to send an alert, a delay is chosen within the interval $[0, 8]$ via the select statement $i : int[0, 8]$. The chosen value is assigned to the *alertDelay* variable. The node then waits in the location **SendAlert** for the duration of the delay and performs carrier sense prior to sending the alert message. This is represented by the edge with the guard function canListen(i), where the node synchronises to the broadcast channel *ALERT[i]* used by other nodes in the vicinity (listening range) to skip sending a message. An MSC showing the alert message transmission is shown in Fig. 11.

We use a representative carrier sense mechanism in the model. Nodes skip sending an alert when another node within their listening range is sending with the same delay. In reality, carrier sense would involve listening to the channel for a small duration before sending the packets. Also, in a case where two nodes start carrier sense at the same instant, their packets would collide since they would start sending at the same instant after the carrier sense delay. In the carrier sense mechanism presented here, we represent the situation in which two nodes can hear each other and have the same delay by one of the two nodes skipping the sending the alert message. When the nodes do not hear each other and the receiver can hear both, the packets collide at the receiver. An example message sequence of carrier sense is shown in Fig. 12. In this example, Node(5) and Node(6) are trying to send alert with the same delay (3ms) as detailed in the execution trace shown in List. 1. In the listing, the

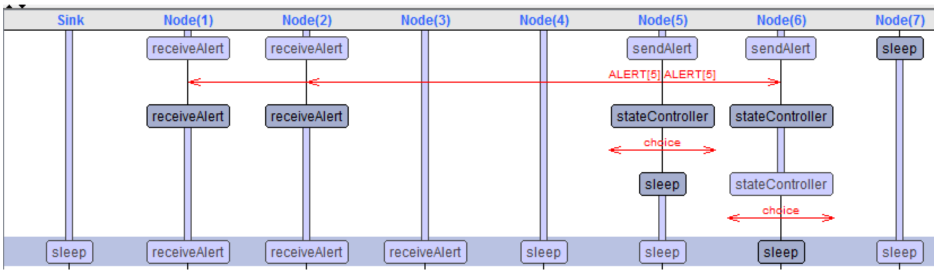


Figure 12: Message sequence chart for carrier sense during alert

Listing 1: Carrier sense execution trace

```
(Sleep, receiveAlert,ReceiveAlert,ReceiveAlert,Sleep,
        StateController,StateController,Sleep)
choice: Node(5)[][3]->  // Delay of 3 ms chosen by node 5
(Sleep,ReceiveAlert,ReceiveAlert,ReceiveAlert,Sleep,
       SendAlert,StateController,Sleep)
choice: Node(6)[][3]->  // Delay of 3 ms chosen by node 6
(Sleep,ReceiveAlert,ReceiveAlert,ReceiveAlert,
       Sleep,SendAlert,SendAlert,Sleep)
ALERT[5]: Node(5)-> Node(1)[5]Node(2)[5]Node(6)[5]
// Alert send by node 5 is heard by node 2 and node 6
(Sleep,ReceiveAlert,ReceiveAlert,ReceiveAlert,
       Sleep,StateController,StateController,Sleep)
```

eight-tuples indicate the states of the nodes starting with the sink followed by nodes 1-7 for the execution related node topology shown later in Fig. 16. We have added comments with prefix “//" to add more detail. When Node(5) begins to send the alert, Node(6) senses the sending and skips sending alert via the edge guarded by canListen(i) function.

In a case where the channel is free, the nodes send an alert at the time instant after the chosen delay. The sending is represented using the send part of the broadcast channel variable *ALERT[id]!*. The local variable *currentSlot* is updated, along with the variable *sentAlert*, and function updateRecord(). The variable *sentAlert* is used by the node to remember that it has sent an alert. In a case where no superframe change occurs after an alert was sent, a node updates its local variable *savedAlert*. The updateRecord() function updates a global array variable *alertTimeRecord[]* which stores the delay chosen by each node in the given round. This is used to check if a collision has occurred. In certain cases when the alert messages fail to reach the sink due to collision, the *savedAlert* variable is used to save the alert, that is sent again in the next round. During this, the probability edge is not used. Instead, the nodes directly move to the location **SendAlert** via the edge (solid line) with the guard $(alertReceived || savedAlert)$. The variable *alertReceived* represents the case when nodes have to forward an alert received from other nodes towards the sink. The nodes then choose a new delay value from the interval $[0, 8]$ for sending the alert message again.

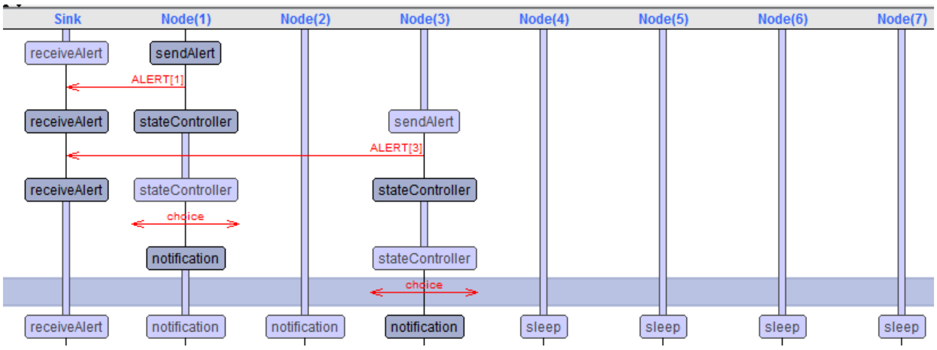


Figure 13: Message sequence chart for collision at the sink

3.4 Collision Modelling

We use a simple collision model similar to the one used in [7] and [6]. In the LMAC [6] protocol, when two nodes send a packet in the same slot, it is considered as a collision. In the DMAMAC protocol model, collision is counted when a node receives at least two alert messages with the same delay in the same alert slot. In our model, we assume that apart from its child nodes, the parents can also listen to nodes in the vicinity which is similar to real networks. We define statically which other nodes a given node can listen to. Based on the representative carrier sense model, the collision occurs at a node only when it receives two alert messages from nodes of the same rank that cannot listen to each other, and had chosen the same delay within the alert slot. A message sequence chart showing a collision occurrence at the sink is shown in Fig. 13.

The execution trace corresponding to the MSC in Fig. 13 is shown in List. 2. In the considered scenario, Node(1) and Node(3) independently choose the same delay of 7 (ms). Since they cannot listen to each other their alert packets end up colliding at the sink. This prevents a change of the superframe (mode of operation). Node(1) and Node(3) detect this and save the alert. The saved alert is used to resend the alert in the next round (with a new delay) to make sure the superframe changes. Note that there could be a situation where collision occurs at lower levels (and even at the sink), but still the change of superframe occurs because of another alert message reaching the sink. For our model, we have created the topology in such a way that both cases exist in different configurations as discussed in Sect. 5. In reality, the receiver nodes do not detect collision: in certain cases nodes receive parts of packets that collide (difficult to decode them) and in other cases they receive nothing at all. In that respect, we rely on a representative model of collision detection designed to be consistent with the effects of collision on the change of superframe.

3.5 DMAMAC-TDMA Node Model

The difference between the Uppaal model for DMAMAC-Hybrid and DMAMAC-TDMA is the alert message mechanism for the nodes. Fig. 14 shows the node model for the TDMA variant of DMAMAC which has a modified alert message mechanism. The model differs from the DMAMAC-Hybrid model in that the alert delay and the listening edge for CSMA

Listing 2: Collision execution trace

```
(ReceiveAlert,StateController,Notification,StateController,
           Sleep,Sleep,Sleep,Sleep)
choice: Node(1)[][7]-> // Delay of 7 ms chosen by node 1
(ReceiveAlert,SendAlert,Notification,StateController,Sleep,
           Sleep,Sleep,Sleep)
choice: Node(3)[][7]-> // Delay of 7 ms chosen by node 3
(ReceiveAlert,SendAlert,Notification,SendAlert,Sleep,
           Sleep,Sleep,Sleep)
ALERT[1]: Node(1)-> Sink[1] // Alert sent by node 1 to the sink
(ReceiveAlert,StateController,Notification,SendAlert,Sleep,
           Sleep,Sleep,Sleep)
ALERT[3]: Node(3)-> Sink[3] // Alert sent by node 3 to the sink
(ReceiveAlert,StateController,Notification,StateController,Sleep,
           Sleep,Sleep,Sleep)
```

(when other nodes send alert at the same time) are eliminated. The +1 on the edge from SendAlert is a representative delay for alert communication. The outgoing edge from ReceiveAlert now has a new update function checkAlertReceived() which replaces the earlier function used to check for collision and update on alerts received. This is also true for the sink model in the TDMA variant, but since this is the only difference between the sink model for DMAMAC-Hybrid and TDMA variant, we do not present the sink model for DMAMAC-TDMA in detail.

The TDMA model has less non-determinism as it does not contain any alert delay, and every node is assigned an exclusive alert slot. Thus, no collision is possible in the network under consideration. In relation to validation and verification, all queries except the queries concerning collision are used to validate and verify also the DMAMAC-TDMA model. An MSC used to validate the proper operation of the TDMA alert procedure is shown in Fig. 15. Note that unlike in DMAMAC-Hybrid alert shown in Fig. 11, all other nodes are in sleep state here. In the DMAMAC-Hybrid variant, nodes send and receive alerts based on their rank in the tree topology.

3.6 Network Topology

The node topology used for the verification of the models is shown in Fig. 16. We use 5 sensor nodes, 2 sensor-actuator nodes, and a sink in the tree topology considered. We consider a small topology to keep the state-space small which is needed in order to conduct exhaustive verification. The current node topology has 3 ranks but since the sink only listens (and does not send alerts), we have 2 alert slots in the DMAMAC protocol. This representative topology allows for both carrier sense and collision, has both sensors and actuators with data communication for both types of nodes, and also has multiple hops. A topology based assumption for listening range is that a lower rank node is listening to all of its children nodes, and also sometimes to other nodes in the vicinity. A real node has a listening range based on its receiver sensitivity, and distance with other nodes in the vicinity that varies

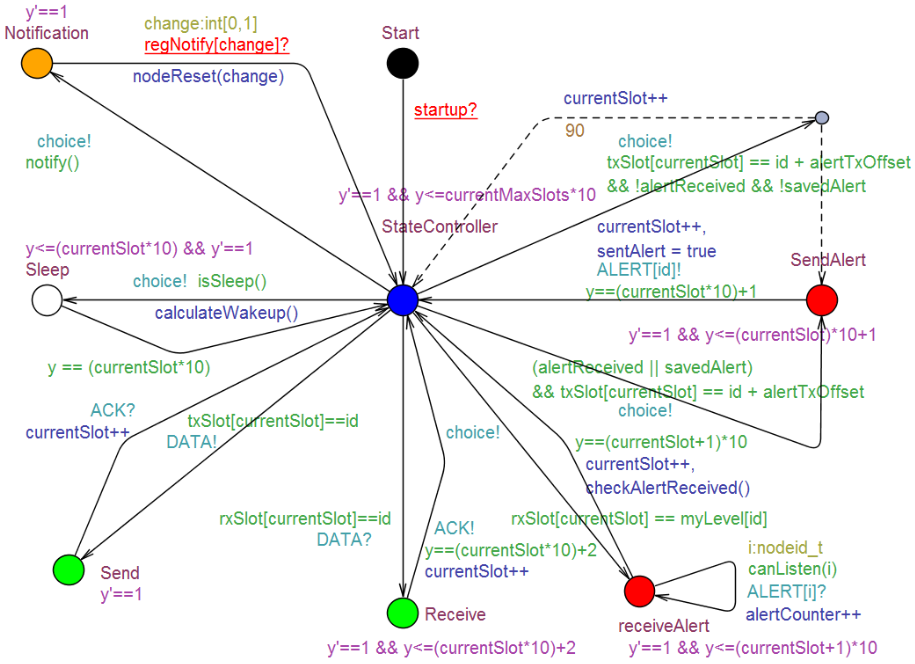


Figure 14: Timed automata model for DMAMAC-TDMA nodes

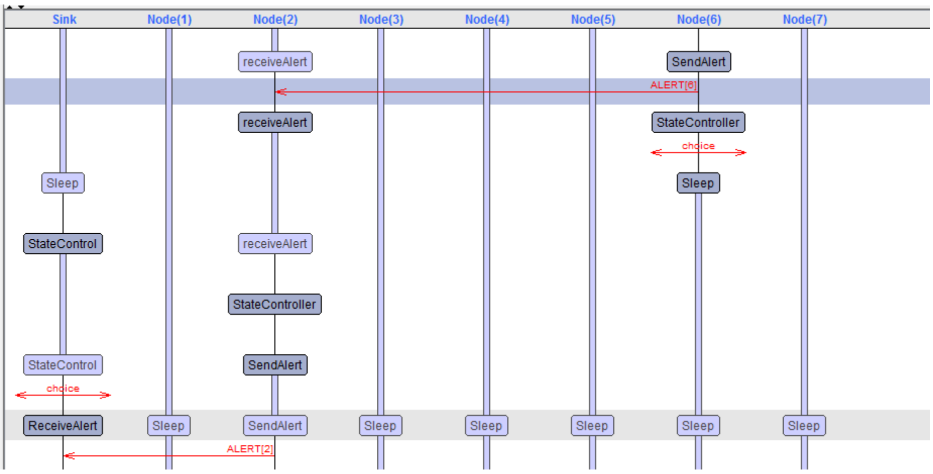


Figure 15: Message sequence chart illustrating alert messaging in DMAMAC-TDMA

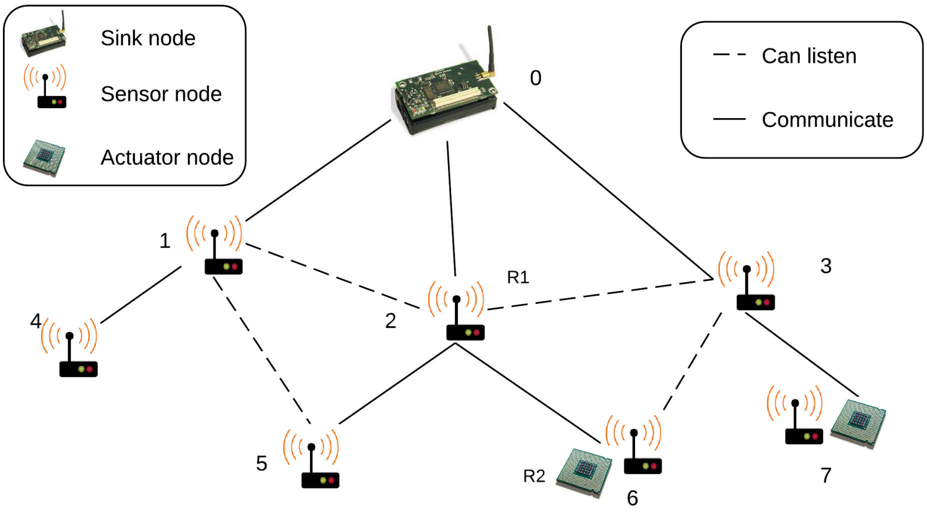


Figure 16: Node topology scenario used for verification

with topology. Given that our main aim is to check the working of the protocol, we define the listening range in the topology manually instead of calculating it dynamically based on multiple factors like node position, path-loss, and receiver sensitivity as is typically done in network simulators for quantitative analysis.

In the topology used for the evaluation of the DMAMAC protocol, the functionalities that need to be verified are covered. The DMAMAC protocol is used for applications with offline scheduling. This means that scheduling is done prior to deployment and all slot allocations are known prior to deployment. The topology in general is pre-planned, and no random deployment is used. A real topology would be much larger than the one considered here. In the current topology, we have 3 levels and a maximum of 2 hops. For a qualitative analysis this covers the error scenarios that could potentially exist with multiple-hops.

The schedule for the considered node topology and for execution of DMAMAC-Hybrid is shown in Fig. 17. The schedule for DMAMAC-TDMA is shown in Fig. 18. These schedules show both sender/receiver identification (node/sink). The sending part is marked by TX and receiving part is marked by RX. For notification messages, only the sender identification (sink) is marked. The schedule only represents the steady superframe. In the transient superframe only the first *Nt* part is used with the alert parts replaced by sleep. The alert parts for DMAMAC-Hybrid are marked with levels/ranks (R#) and for DMAMAC-TDMA with the node identity. Note that we use 10 milliseconds (“ms") as slot duration similar to slot sizes used in general practise.

3.7 Configurations

Multiple configurations of the DMAMAC protocol can be analysed based on values that can be varied in the model. Firstly, the Uppaal model can start in either steady or transient mode and this could have an effect on some of the verification properties (as discussed in

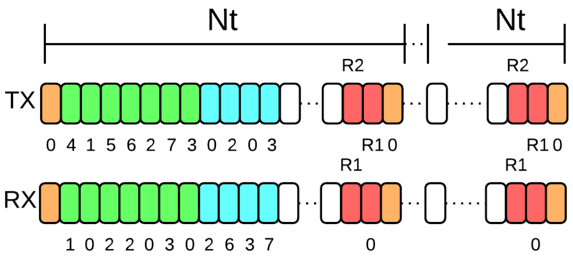


Figure 17: Superframe structure for DMAMAC-Hybrid protocol execution

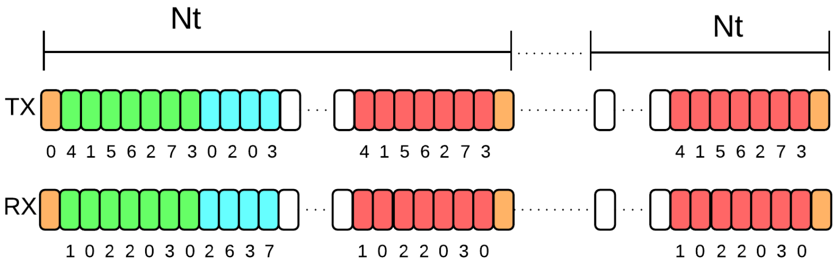


Figure 18: Superframe structure for DMAMAC-TDMA protocol execution

Sect. 5). Another important factor affecting the configurations for DMAMAC-Hybrid is the range of values for the variable *alertDelay*. In the protocol, we have used the range [0,8] to reduce collision. Due to state-space explosion, we use only $alertDelay[1,1]$ for exhaustive queries, e.g., deadlock query. The $alertDelay[1,1]$ in itself covers all possibilities including possibility of state-switch, collision and CSMA, and hence all the qualitative aspects of the protocol. For other non-exhaustive queries, we use up to $alertDelay[1,4]$ configurations to further verify the protocol. The only difference between $alertDelay[1,1]$ and the other considered configurations is the applicability of property *sink mode* and *consistent node mode* of the verification properties and is further discussed in Sect. 5. Also, the select statement interval $[1,10]$ used to decide the switch from transient to steady mode by the sink is reduced to $[1,2]$ to keep the state-space small for all the queries. The reduction of the interval only means that in transient mode there is a 50% probability to switch to steady mode, and thus does not affect the qualitative results. The DMAMAC-TDMA model in general does not require any configuration changes similar to DMAMAC-Hybrid to prevent state-space explosion.

4 Model Validation

We first validate the constructed Uppaal models of the DMAMAC protocol by checking some basic behavioural properties related to the operation of the model. The purpose is to obtain a high degree of confidence in the constructed model prior to verifying key properties of the protocol in the next section. During construction of the DMAMAC protocol, we validated the operation of the model via MSCs obtained from step-by-step execution of the model in the Uppaal simulator as illustrated in the previous section. These properties were related to data transmission between nodes, data transmission between the nodes and the sink, sending/receiving of alert message, possibility of collision, and carrier sense when sending alert messages.

Below we validate properties of the model related to data communication and collisions using the verification engine of Uppaal. For this, we express the properties to be validated in the form of Uppaal queries. Queries in Uppaal are written in a restricted variant of Computation Tree Logic (CTL) in which path formulas cannot be nested. Specifically, the following path formulae are supported by Uppaal: A$\Box$ (always globally), A$\Diamond$ (always eventually), E$\Diamond$ (reachable), and E$\Box$ (exists globally).

For validation purposes, we first check the operation of the model with respect to data communication between neighbouring nodes and between the sink and its neighbouring nodes. We check that if two nodes i and j are such that the parent node of node j is i, then these will eventually communicate. Furthermore, it should be such that any child node of the sink node should eventually communicate with the sink. Formally, these two properties can be expressed as the set of queries below. Here, $nodeid\_t$ is the type used to represent node identifiers in the model, $parent[i]$ is used to obtain the parent node of node i, and $sinkId$ denotes the identity of the sink. The property is expressed by reference to the location **Sent** and location **Received** which are reached by the communicating nodes upon synchronization over the channel *DATA*.

Node data communication
$\forall$ i,j $\in$ nodeid_t such that parent[j]==i: A$\Diamond$ (Node(i).Sent $\land$ Node(j).Received)

Sink data communication
$\forall$ i $\in$ nodeid_t such that parent[i]==sinkId: A$\Diamond$ (Node(i).Sent $\land$ Sink.Received)

It should be noted that we do not check the property that two neighbouring nodes always have the possibility to communicate. This is due to the fact that Uppaal does not support nesting of CTL path formulae.

The second property that we validate is related to collisions which play an important role in the DMAMAC protocol in relation to the sending of alert messages. In this case, we check that it is possible to have collision happening on all nodes and on the sink. Collision cannot be guaranteed to happen and hence we verify only the possibility of collision occurring. Formally, these two properties are expressed as the following set of queries:

Node collisions $\forall$ i $\in$ nodeid_t : E$\Diamond$ Node(i).collision

Sink collisions E$\Diamond$ Sink.collision

Finally, we also validate that there are no deadlocks in the model. In Uppaal, this can be expressed via the query below where *deadlock* is a built-in state property in Uppaal.

No deadlock A$\Box$!deadlock

The above queries related to data communication, collision, and deadlocks were all satifsied on both the transient and the steady variant of the DMAMAC-Hybrid model. The DMAMAC-TDMA model is validated against queries for data communication and deadlock as collision is not a possibility for DMAMAC-TDMA model.

5 Protocol Verification

We now consider verification of the key functional properties of the DMAMAC protocol. As explained earlier, the constructed model comes in two variants: one variant with the protocol starting in the transient mode and one variant with the protocol starting in the steady mode. We first consider common properties that are independent of whether the protocol starts in the transient or in the steady mode. Then we consider properties specific for the transient mode case followed by properties specific for the steady mode case. Finally, we verify two real-time properties of the protocol related to upper bounds on mode switch delay and data transmission delay.

5.1 Common Properties

Given the dual-mode operation of the DMAMAC protocol, the important properties relate to the nodes operating in different modes, and switching between them. Firstly, we check the operating mode properties. We make sure that the sink is exclusively either in the steady mode or in the transient mode at all times. Following this, we check that all nodes follow the operating mode of the sink consistently. Formally, these properties are expressed as follows:

Sink mode A□ (Sink.steady ∧ !Sink.transient) || (!Sink.Steady ∧ Sink.transient)

Consistent node mode ∀ i ∈ nodeid_t:
A□ (Node(i).transient == Sink.transient || Node(i).steady == Sink.steady)

Next, we investigate properties of the protocol related to collision and its effect on the change of operational modes. The queries refer to the *changeSuperframe* variable which indicates whether the network should change mode in the next superframe. Collisions may have different effects depending on the configuration under consideration. For configurations with $alertDelay[1,1]$ where all the nodes will pick the same delay, collision at the sink should not result in a change of superframe or operational modes. For configuration with $alertDelay[1,2]$, we may have both collision and change of superframe since, e.g., two nodes may pick a delay of 1 (which will result in a collision) while a single third node picks a delay of 2. The latter choice will result in the sink being notified of a required change of mode. Formally, properties related to collisions and change of mode are specified as follows:

Collision and mode switch E◇(Sink.collision ∧ Sink.changeSuperframe)

Collision and no mode switch E◇(Sink.collision ∧ !Sink.changeSuperframe)

Following the discussion above, we expect that the first property is false in configurations where all nodes must choose the same delay while it is true in configurations where different alert delays can be chosen. This implies that the protocol design ensures that the DMAMAC protocol can change mode even in the presence of collisions. The second property is expected to be true as we may (in all configurations) have the situation that the choice of delay (alert) causes collisions such that the sink may not be notified of the required change of mode in the current superframe. Of course, the sink may be notified via retransmission of the alert in a later superframe, eventually causing a mode switch (see below).

Listing 3: Simultaneous collision and change of superframe

```
(ReceiveAlert,StateController,StateController,StateController,
 Sleep,Sleep,Sleep,Sleep)
choice: Node(1)[][1]-> // Node 1 chose delay of 1 ms
(ReceiveAlert,SendAlert,StateController,StateController,Sleep,
 Sleep,Sleep,Sleep)
choice: Node(3)[][1]-> // Node 3 chose delay of 1 ms
(ReceiveAlert,SendAlert,StateController,SendAlert,Sleep,
 Sleep,Sleep,Sleep)
choice: Node(2)[][2]-> // Node 2 chose delay of 2 ms
(ReceiveAlert,SendAlert,SendAlert,SendAlert,Sleep,
 Sleep,Sleep,Sleep)
ALERT[3]: Node(3)->Sink[3] // Node 3 sends alert
(ReceiveAlert,SendAlert,SendAlert,StateController,Sleep,
 Sleep,Sleep,Sleep)
ALERT[1]: Node(1)->Sink[1] // Node 1 sends alert
(ReceiveAlert,StateController,SendAlert,StateController,Sleep,
 Sleep,Sleep,Sleep)
ALERT[2]: Node(2)->Sink[2] // Node 2 sends alert with a higher delay (2 ms)
(ReceiveAlert,StateController,StateController,StateController,Sleep,
 Sleep,Sleep,Sleep)
```

An example trace demonstrating co-existence of collisions and change of superframe is shown in List. 3. In this example, the nodes 1 and 3 choose the same delay (1 ms) and cannot listen to each other. The transmissions therefore collide at the sink. But node 2 which has chosen a different delay (2 ms) successfully alerts the sink thus inducing change of superframe. This delay choice is done when the model changes location from **StateController** to **SendAlert**.

Finally, we verify a property related to the critical change of state in the protocol from steady to transient. When the data requirement is higher, the protocol should be able to detect and switch accordingly. Also, when a switch fails due to collisions, there should be a possibility to re-use the failed alert to induce change of operational modes. The failed alert is used as a saved alert in the next alert round. We use the query which searches for one example where this occurs.

Critical change of state $\exists$ i $\in$ nodeid_t:
E$\Diamond$ Node(i).savedAlert $\wedge$ Sink.steady $\wedge$ Sink.changeSuperframe

It should be noted that given the nature of the model, the collisions could occur forever preventing the change of superframe. This means that we cannot show that a state-switch will eventually happen.

5.2 Transient Model Variant

We now consider properties specific for the variant of the model where the sink and nodes starts in transient mode. In the model, transient and steady are boolean variables. In the case where the controller process stays in the transient state permanently, the protocol needs to stay in transient mode of operation to suit the application needs. The query below checks if there is a path where the system invariantly is in the transient mode.

Remain transient $E\Box$ transient

The second property represents the reachability of steady mode from the starting state, i.e., that it is possible for the system to change mode from transient to steady.

Steady switch $E\Diamond$ steady

5.3 Steady Model Variant

For the variant of the model that starts in the steady mode, we verify the dual properties of the variant that starts in the transient mode. These two properties are listed below:

Remain steady $E\Box$ steady

Transient switch $E\Diamond$ transient

The two properties check that it is possible to remain in steady mode and that it is possible to switch to transient mode.

5.4 Real-time Properties

We now consider real-time properties related to mode switch delay and data communication delay. In order to verify these properties, we use a modified version of the Uppaal model where we have included the use of two watch templates (Watch1 and Watch2) in order to record elapsed time.

5.4.1 Switch Delay

The first real-time property that we consider is the switch delay, i.e., the time difference between a detection of threshold breach and the mode switch happening. This switch is required to happen within the duration of a superframe (transient superframe length). This property is specified as follow:

Switch delay $A\Box$ Watch1.switchDelay $\leq$ superframeLength

We verified the switch delay property by considering a single node farthest from the sink. By symmetry, the property applies to other nodes at the same rank, and also to the parent nodes which (by the tree topology) will have a smaller maximum switch delay.

Model Extension for Verifying Switch Delay.

For verification of switch delay an extension to the existing Uppaal model is required. Firstly, we have created an extra template to measure the switch delay from a given node which is required in order to verify the switch delay of DMAMAC. The timer has two locations mainly tracking the change of states based on the synchronisation channels START and STOP as shown in Fig. 19.

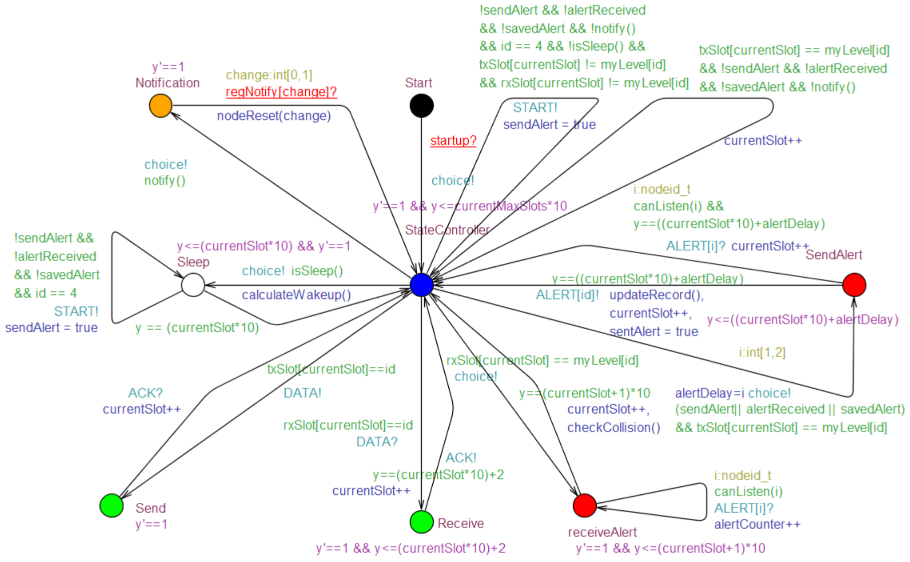


Figure 20: Node model used in the switch delay verification

The change of state in consideration is steady to transient based on an alert message. Also, the existing node model is modified to measure possible switch delays or worst case switch delay per node basis using the query language of Uppaal. The sink model is only extended to incorporate a stop timer. The initiation of an alert message occurs when the measured readings from the sensor exceeds a given threshold level [3]. This can happen at any given time. Thus, we consider mainly the possibility of alert message generation when the radio is in sleep mode or is intermediate state for state transitions. If in the data mode, the sensory readings are sent directly, which is measured by the sink directly and can implement a superframe change if a threshold breach is detected. The alternative node model for the switch delay verification is shown in Fig. 20.

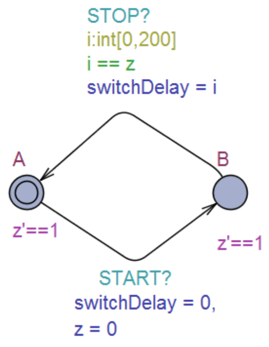


Figure 19: Switch delay timer

The stop channel synchroniser is placed in the sink model. The model checks for a change of superframe from steady to transient. As shown in Fig. 21, an edge is placed in the Notification location to check for superframe change. This can only occur due to a generated alert. In the background, it is verified that it has been generated from the same node in consideration (for proper per node delay result). We statically design node with $Id == 4$ to generate alert at random time and induce a state switch. Network wide synchronisation in a real case scenario might vary depending on packet loss effecting the notification packets sent by the sink. In our case we do not consider packet loss. The current switch delay

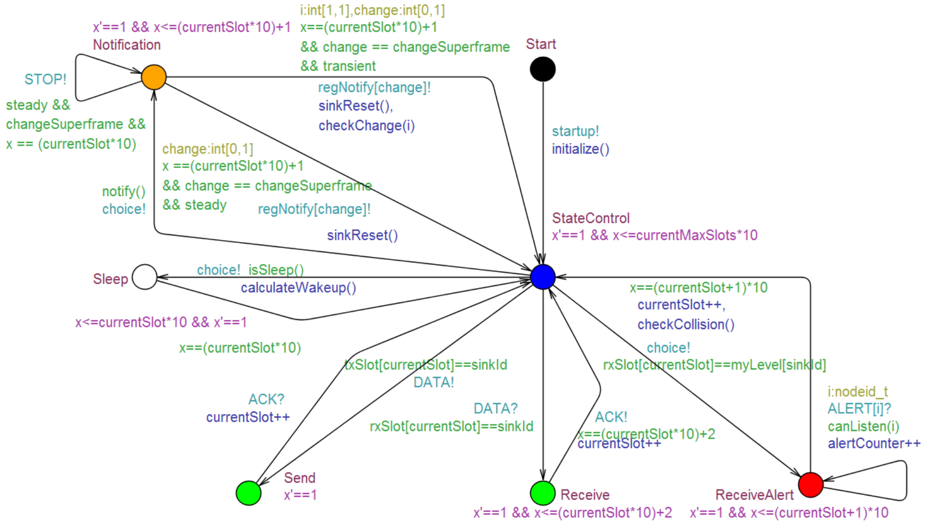


Figure 21: Sink model used for switch delay verification

model was designed for DMAMAC-Hybrid. A similar extended model was designed for DMAMAC-TDMA.

5.4.2 *Data delay*

The second real-time property concerns the data communication delay. It is the time elapsed between the first data sent in the superframe until the last data received. This is required to be within the same superframe. The property is expressed as follows:

Data delay A□ Watch2.dataDelay ≤ superframeLength

Model Extension for Verifying Data Delay

In other words data communication delay it is the time required for all data transmission within one superframe. Similar to the switch delay a model extension is made to the existing Uppaal model. An extra template is used to measure the data delay when conducting verification. The template is shown in Fig. 22 and contains 3 locations, an extra location (urgent) is used due to constraints imposed by the Uppaal model in terms of modelling. Otherwise, the data delay timer functions similar to the switch delay timer using channel synchronisation variables STARTER and STOPPER.

The Uppaal node model was extended to include the channel synchronisation variables at Send and Receive locations as shown in Fig. 23. Based on static scheduling shown in Fig. 17, the first node to send data is node 4, and the last node to receive data is node 7. Given the absence of packet failure and re-transmission we can

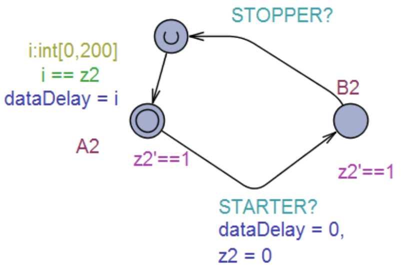


Figure 22: Data delay timer

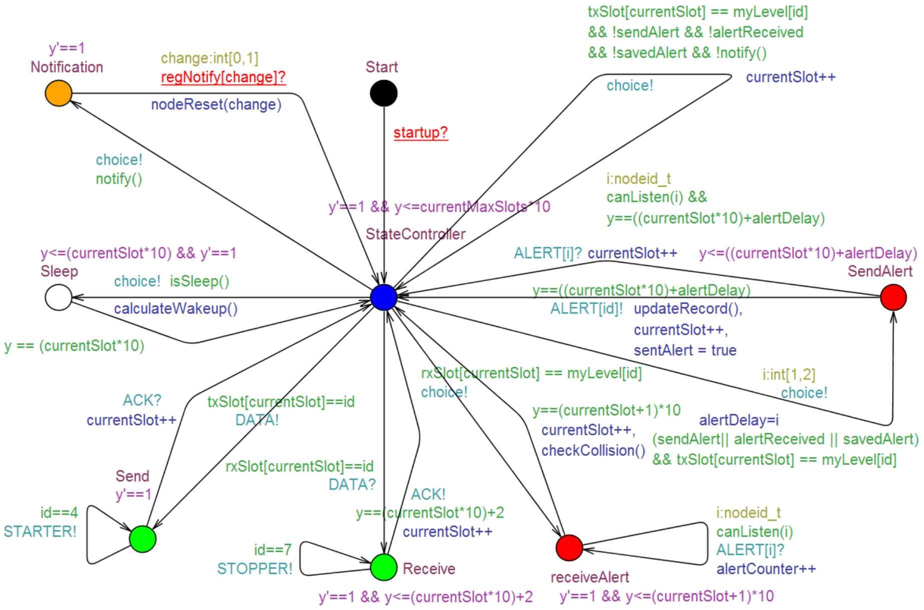


Figure 23: Node model used for for data delay verification

assume that the data received by node 7 is the last data packet in the superframe. The current data delay model was designed for DMAMAC-Hybrid. A similar model extension was also developed for the DMAMAC-TDMA variant.

5.5 Discussion

All properties listed above evaluate to the expected results. The properties related to collisions were only considered on for hybrid variant of DMAMAC since collisions are not possible in DMAMAC-TDMA. Details on the execution time for the queries on different configurations of the model are shown in table 1 for the hybrid version and in table 2 for DMAMAC. The verification was conducted on a PC with 4 GB RAM, 2.30 GHz 2-core processor. The query **Collision and mode switch** could be verified only on the configuration with $alertDelay[1,1]$ and resulted in memory exhaust in other configurations. Other queries were verified also on configuration $alertDelay[1,2]$, and $alertDelay[1,4]$ with $i : int[1,2]$.

6 Validation of DMAMAC Simulation Model and Implementation

In earlier work [4] we have developed a simulation model of DMAMAC on the OMNeT++-MiXiM [9] platform. The purpose of this simulation model was to assess the

Property / Query	Result	CPU Time (s)	Resident Mem. (KB)	Virtual Mem. (KB)
Configuration : alertDelay[1,1], i:[1,2]				
Common queries				
Sink mode	Not Satisfied	718.837	1,795,596	3,595,148
Consistent node mode	Satisfied	876.586	1,772,436	3,573,820
Collision and mode switch	Not Satisfied	711.287	1,797,488	3,599,136
Collision and no mode switch	Satisfied	3.214	21,044	49,512
Critical change of state	Satisfied	33.868	113,084	238,116
Transient specific queries				
Remain transient	Satisfied	0.015	13,820	56,924
Steady switch	Satisfied	0.64	13,888	40,168
Steady specific queries				
Remain steady	Satisfied	0.032	17,424	58,892
Transient switch	Satisfied	1.965	19,308	48,796
Real-time queries				
Switch delay	Satisfied	273.048	440,676	891,152
Data delay	Satisfied	231.302	450,664	905,284
Configuration: alertDelay[1,2], i:[1,2]				
Common queries				
Sink mode	N/A	N/A	Memory exhausted	Memory exhausted
Consistent node mode	N/A	N/A	Memory exhausted	Memory exhausted
Collision and mode switch	Satisfied	8.331	62,292	128,008
Collision and no mode switch	Satisfied	8.346	61,616	129,064
Critical change of state	N/A	N/A	Memory exhausted	Memory exhausted
Transient specific queries				
Remain transient	Satisfied	0.02	13,836	40,092
Steady switch	Satisfied	0.562	13,848	57,012
Steady specific queries				
Remain steady	Satisfied	0.046	36,596	82,700
Transient switch	Satisfied	16.708	66,116	137,096
Real-time queries				
Switch delay	Satisfied	1200.225	1,799,596	3,601,164
Data delay	Satisfied	1068.014	1,799,624	3,622,604

Table 1 Verification results for DMAMAC-Hybrid

performance of the protocol on larger network configuration taking into account additional network parameters such as packet transmission failures. Recently, we have also made an implementation of DMAMAC for the TinyOS operating system [10] running on Zolertia Z1 platform [15]. The primary purpose of the prototype implementation was to check implementability of the protocol and perform a deployment testing in a real physical environment. In particular to check that the underlying assumptions of the protocol design are sound from the perspective of practical use.

The simulation model and the implementation was developed manually with reference to the specification of the protocol presented in the earlier version [11] of the present paper. In order to further validate that the simulation model and the implementation conforms to the formal specification presented in this paper, we compare execution sequences from the

Property / Query	Result	CPU Time (s)	Resident Mem. (KB)	Virtual Mem. (KB)
Configuration : i:[1,2]				
Common queries				
Sink mode	Not Satisfied	587.079	616,520	1,231,784
Consistent node mode	Satisfied	618.902	611,156	1,218,792
Critical change of state	Satisfied	33.868	113,084	238,116
Transient specific queries				
Remain transient	Satisfied	0.016	15,036	59,036
Steady switch	Satisfied	0.843	15,012	59,036
Steady specific queries				
Remain steady	Satisfied	8.783	16,056	60,508
Transient switch	Satisfied	12.015	28,572	68,324
Real-time queries				
Switch delay	Satisfied	1,934.927	1,958,796	3,923,496
Data delay	Satisfied	2.278	14,536	39,756

Table 2 Verification results for DMAMAC-TDMA

Uppaal model with the simulation model execution, and the deployed model execution. The considered execution sequences conforms to the superframe structure and requirements specification of the DMAMAC protocol. The details of the sequences varies from platform to platform but when we abstract from the platform specific detail we can check that execution performed by the simulation model and the implementation is also allowed by the Uppaal model, i.e., conforms to the specification. Furthermore, this approach does not require a an exhaustive state-space search which allow us to investigate larger configurations.

An example of a sequence diagram obtained from Uppaal is shown in Fig. 24, an example of a sequence diagram from MiXiM simulation is shown in Fig. 25, and the sequence diagram obtained from TinyOS is shown in Fig. 26. In this case, we have extracted sequence diagram from data forwarding towards sink from Node 3 via Node 1. The sequence diagrams for MiXiM and TinyOS are obtained from the log files created during the executions. When we abstract away the platform-specific details from the sequence diagrams obtained from MiXiM and from TinyOS implementation, it can be seen that the three diagram represents the same exchange of messages and hence for the considered scenario, the simulation model and the implementation conforms to the protocol as specified by the Uppaal model.

7 Conclusion and Perspectives

In this article, we have detailed the modelling, validation, and verification process of the DMAMAC protocol. The DMAMAC protocol is designed for process control applications and we have used the Uppaal model-checking tool for modelling and verification. The model consists of a network of timed automata with multiple nodes and a sink operating according to the DMAMAC protocol. The Uppaal models serves as a formal specification for both the hybrid and TDMA variant of DMAMAC. We have explained the model in Uppaal including its modelling elements and templates in detail. The constructed timed automata model includes generic MAC slot operations including data sending and receiving,

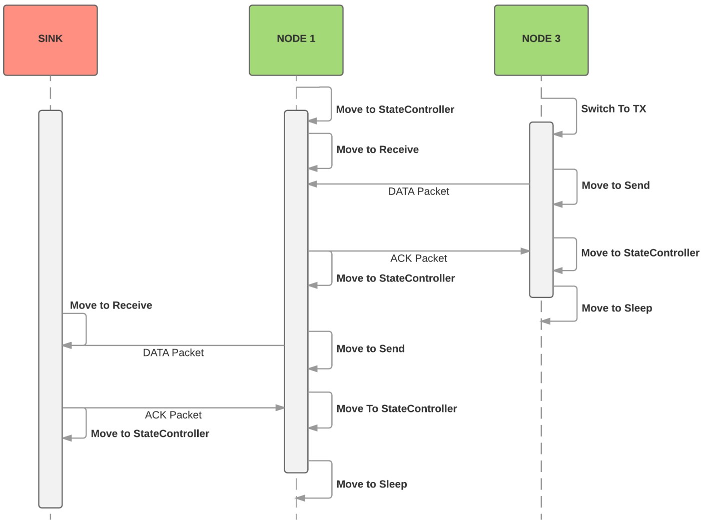


Figure 24: Data transmission sequence diagram from Uppaal model

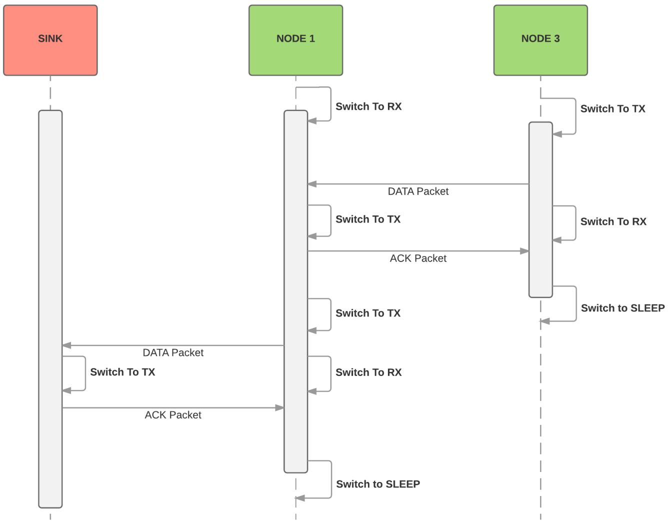


Figure 25: Data transmission sequence diagram for OMNeT simulation model

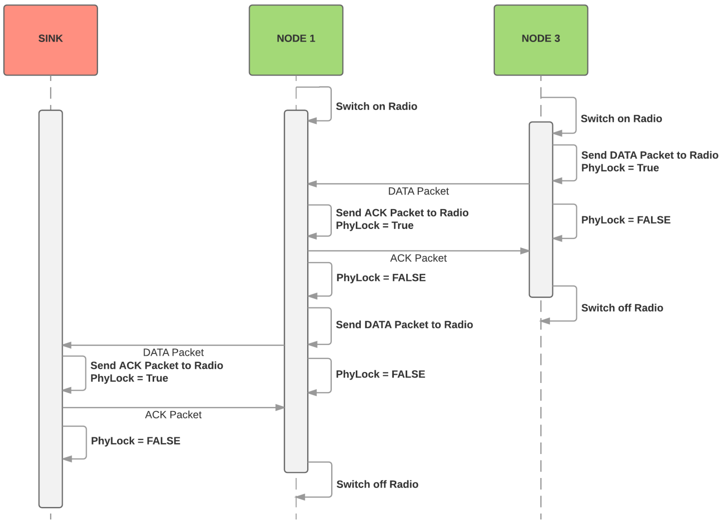


Figure 26: Data transmission sequence diagram from TinyOS implementation

notification, and sleep. This means that the model can be extended to represent other MAC protocols with similar (and extra) slotting within their superframe.

We have validated the basic operation of the constructed model using message sequence charts highlighting the most important features, and operations of the protocol including data transfer, alert message functioning, carrier sense, and possibility of collision. Further, we validated the proper operation of the model using the verification engine of Uppaal. The validated model was then verified for the switch procedure and safety properties, including absence of deadlock and other faulty states. The key real-time properties in the form of upper bounds on switch delay and data delay were verified. Two variants of the model were used for verification and validation, one starting with the transient mode of operation and the other starting with steady mode. Different configurations of the model with varying alert delay were used as a basis for the verification for DMAMAC-Hybrid in particular. For verification, we used a representative node topology that covers all important features of the protocol including existence of sensors and actuators, multi-hop, alert messages, and possibility of collision. Furthermore, we have used execution sequence from the constructed Uppaal models to validate the simulation model of DMAMAC that we have developed for performance analysis of the protocol, and we have used execution sequences from the Uppaal model to validate our prototype implementation of DMAMAC. This in turn increases the confidence in the simulation model and in the DMAMAC implementation.

As a proposed future work, a stochastic model of the DMAMAC protocol to verify the quantitative properties including collision probability, expected switch delay, and energy consumption could provide further insights to the working of the protocol.

References

[1] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless Sensor and Actor Networks: Research challenges. *Ad Hoc Networks*, 2(4):351–367, 2004.

[2] Joao P Hespanha, Payam Naghshtabrizi, and Yonggang Xu. A survey of recent results in Networked Control Systems. In *Proceedings of the IEEE*, volume 95, pages 138–162, 2007.

[3] Admar Ajith Kumar Somappa, Knut Øvsthus, and Lars Michael Kristensen. Towards a dual-mode adaptive MAC protocol (DMA-MAC) for feedback-based networked control systems. *Procedia Computer Science*, 34:505–510, 2014. The 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops.

[4] A. Ajith Kumar Somappa, Lars M. Kristensen, and Knut Ovsthus. Simulation-based evaluation of DMAMAC: A dual-mode adaptive mac protocol for process control. In *Proceedings of the 8th International Conference on Simulation Tools and Techniques*, SIMUTools '15, pages 218–227. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015.

[5] Ansgar Fehnker, Rob Van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann, and Wee Lum Tan. Automated analysis of AODV using Uppaal. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 173–187. Springer Berlin Heidelberg, 2012.

[6] Ansgar Fehnker, Lodewijk van Hoesel, and Angelika Mader. Modelling and Verification of the LMAC protocol for Wireless Ssensor Networks. In *Integrated Formal Methods*, volume 4591 of *Lecture Notes in Computer Science*, pages 253–272. Springer Berlin Heidelberg, 2007.

[7] Simon Tschirner, Liang Xuedong, and Wang Yi. Model-based Validation of QoS properties of Biomedical Sensor Networks. In *Proceedings of the 8th ACM International Conference on Embedded Software*, EMSOFT '08, pages 69–78, New York, NY, USA, 2008. ACM.

[8] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas Van Vliet, and Zheng Wang. Statistical Model Checking for networks of Priced Timed Automata. In *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 6919 of *LNCS*, pages 80–96. Springer Berlin Heidelberg, 2011.

[9] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Klein Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in OMNeT++ the MiXiM vision. In *SIMUTOOLS*, pages 71:1–71:8, 2008.

[10] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In *Ambient Intelligence*, pages 115–148. Springer, 2005.

[11] Admar Ajith Kumar Somappa, Andreas Prinz, and Lars M Kristensen. Model-based verification of the DMAMAC protocol for real-time process control. *VECoS 2015 Verification and Evaluation of Computer and Communication Systems*, 1431:81–96, 2015.

[12] Petcharat Suriyachai, James Brown, and Utz Roedig. Time-critical data delivery in Wireless Sensor Networks. In *Proceedings of 6th IEEE International Conference on Distributed Computing in Sensor Systems*, volume 6131, pages 216–229, 2010.

[13] Gerd Behrmann, Alexandre David, and KimG. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer Berlin Heidelberg, 2004.

[14] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *SIMUTOOLS*, pages 60:1–60:10, 2008.

[15] Z1 datasheet, zolertia. `http://www.zolertia.io//`. Accessed: 25-01-2016.

Chapter 10

Implementation and Deployment Evaluation of the DMAMAC Protocol for Wireless Sensor Actuator Networks

Available online at www.sciencedirect.com

SciVerse ScienceDirect

ELSEVIER

Procedia Computer Science 00 (2016) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

Implementation and Deployment Evaluation of the DMAMAC Protocol for Wireless Sensor Actuator Networks

Admar Ajith Kumar Somappa*[a,b], Knut Øvsthus[a], Lars Michael Kristensen[a]

[a]*Faculty of Engineering and Business Administration, Bergen University College, Norway*
[b]*Faculty of Engineering and Science, University of Agder, Norway*

Abstract

The increased application of wireless technologies including Wireless Sensor Actuator Networks (WSAN) in industry has given rise to a plethora of protocol designs. These designs target metrics ranging from energy efficiency to real-time constraints. Protocol design typically starts with a requirements specification, and continues with analytic and model-based simulation analysis. State-of-the-art network simulators provide extensive physical environment emulation, but still has limitations due to model abstractions. Deployment testing on actual hardware is therefore vital in order to validate implementability and usability in the real environment. The contribution of this article is a deployment testing of the DMAMAC protocol. DMAMAC is an energy efficient protocol recently proposed for real-time process control applications and is based on Time Division Multiple Access (TDMA) in conjunction with dual-mode operation. A main challenge in implementing DMAMAC is the use of a dynamic superframe structure. We have successfully implemented the protocol on the Zolertia Z1 platform using TinyOS (2x). Our scenario-based evaluation shows minimal packet loss and smooth mode-switch operation, thus indicating a reliable implementation of the DMAMAC protocol.

Keywords: Wireless sensor networks and applications, Network architecture and design, Communication protocols.

1. Introduction

Wireless Sensor and Actuator Networks (WSANs)[1] consist of interconnected sensors and actuators collaborating to perform monitoring and control. The sensors measure physical quantities and report it to the actuators which act on them. WSANs generally use a powerful central sink to collect and process the collected data and send commands to the actuators. The sink also handles network functions such as routing and scheduling of medium access. WSANs are applied across multiple domains including factory and process automation, healthcare, smart homes, and smart buildings. Energy efficiency and real-time constraints constitute key requirement for WSAN protocol designs.

[*] Corresponding author. Tel.: +47 55 58 75 88
E-mail address: aaks@hib.no

Medium Access Control (MAC) protocols control the network wide medium access for the sensor and actuator nodes. MAC protocols directly control the communication and is thus accessing the radio which in turn is the largest consumer of energy. The Dual-Mode Adaptive MAC (DMAMAC) protocol[2] is an energy efficient MAC protocol proposed for real-time process control. Process control applications in the context of DMAMAC have two important states of operation: steady and transient state. The steady state corresponds to periods with minimal change in measured physical quantities. The transient state corresponds to periods with substantial change in measured data. The process control alternates between these two states throughout the operation of the system. The dual-mode operation of DMAMAC reflects the dual states in process control applications. The transient mode of DMAMAC has high data rates for transient state operations. The steady mode offers lower data rate and energy conservation for steady state operation. This makes the protocol energy efficient while still being able to satisfy real-time requirements.

The related single-mode GinMAC[3] protocol has been proposed for process monitoring and control applications. The design of the DMAMAC transient superframe is based on the GinMAC protocol features. An implementation of the GinMAC protocol on TinyOS was tested with a network of 15 nodes in a tree topology[4]. The implementation was used to evaluate the performance of the GinMAC protocol via deployment testing. The TKN15.4 package is one of the important MAC implementations on TinyOS[5] developed by Technical University of Berlin. Prime features of TKN15.4 include platform-independence and modularity based on an adaptation of the IEEE 802.15.4-2006 standard. MAC implementations are also provided via the alternate MAC Layer Architecture[6].

The main contribution of this article is to demonstrate the practical implementability of the newly proposed DMAMAC protocol on the Zolertia Z1[7] platform with CC2420 transceivers (IEEE 802.15.4) running TinyOS[8]. Also, we deploy the implementation in a factory automation context. TinyOS is a component-based operating system for WSANs using the nesC language to implement applications. A fundamental difference between the related MAC implementations discussed above and the DMAMAC implementation is the realisation of a dynamic MAC superframe. The use of a dynamic superframe entails a network-wide mode-switch procedure, which is the central aspect of the DMAMAC protocol. Furthermore, we also show that DMAMAC can be implemented by relying on off-the shelf components for implementation as plugins, thus increasing re-usability. In particular, this holds for the time synchronisation which is another essential part of the DMAMAC protocol used for the TDMA implementation. Another important contribution is that our implementation serves as a benchmark to analyse the actual performance of DMAMAC on hardware. In particular, it enable us to analyse the effect of packet failure on the mode-switch procedure and identify steps to further reduce the effect. We conduct this analysis using multiple deployment scenarios which including inference with noise produced by motors. This provides important knowledge on the effect of external machines as encountered in process control industries which is one intended application domain for DMAMAC.

Outline. The rest of the paper is organised as follows. Section 2 briefly introduces the DMAMAC protocol. In section 3 we discuss the implementation model on TinyOS of the DMAMAC protocol and its integration with the existing TinyOS infrastructure. In section 4 we present the deployment experiments done and analyse the performance results that were obtained. Finally, in section 5 we sum up the conclusions and discuss future work.

2. The DMAMAC Protocol

DMAMAC-TDMA[9] is a Time Division Multiple Access (TDMA) protocol proposed for real-time process monitoring and control applications. It was designed based on the simulation analysis[9] of the initially proposed DMAMAC-Hybrid[2]. Henceforth all references made to DMAMAC in this article refers to DMAMAC-TDMA. DMAMAC is a dual-mode protocol incorporating two operational modes: *transient mode* and *steady mode*. The transient mode corresponds to the operational specification of transient state in process control[2] in which there is typically a large amount of traffic between sensor and actuators in order to control the process. The steady mode of operation corresponds to the steady state of process control in which only a small amount of data traffic is required between nodes in order to monitor the process. The tree network topology used for the DMAMAC protocol consists of sensors, actuators, and a sink. The sink is a powerful central node used for global management and control of the network. The sensors measure physical entities and report the measurements to the sink. The sink processes this data and transmits commands and processed data to the actuators, which act on it. The main features of the protocol including transient mode, steady mode, and alert message handling and are outlined below.

Transient mode. The MAC superframe used by the protocol depends on the current mode of operation. The transient mode superframe and operation is used when the process being controlled is in the transient state. This type of superframe is comprised of six types of slots: notification slots, sensor data transmission slots, actuator data transmission slots, re-transmission slots (to cover for packet loss), sink processing slots, and inactive (sleep) slots. A detailed description of the superframe structure can be found in[2].

Steady mode. The steady mode superframe is used during the steady mode of operation. The design of the steady mode superframe relies on an extension of the transient superframe in that it is a multiple of the transient superframe with an added alert message slot. The additional alert message slots are present in order to support the switch to transient mode when required by the controlled process. The length of the steady mode superframe is flexible and can vary from 2 × the transient superframe length to more depending on the specific application requirements. In particular, the energy consumption of the protocol can be reduced by increasing the length of the steady mode superframe.

Alert messages. An alert message is used by the sensors to trigger a change in the mode of operation. The sensors make this decision based on a steady state threshold interval set by the sink. When the protocol is operating in steady mode and a sensor reading breaches the current threshold, an alert message is generated and sent to the sink using the alert message slots. The DMAMAC (TDMA) protocol[9] that we implement in this paper has a complete TDMA structure. This means that every sensor has its own alert message slot within the steady superframe which is used to send the alert message towards the sink. The sink acts on the alert message by indicating a change in the mode of operation (steady to transient) for the next superframe. This is done using the notification message slot that follows the alert message slots. The superframe change is implemented in the next superframe throughout the network in order to preserve operational mode synchronisation.

Mode-switching and packet failure. Mode-switch is a safety critical part of the DMAMAC protocol due to its dual mode nature. The mode-switch from steady to transient is initiated by the sensor nodes via alert messages based on a detected threshold breach. The mode-switch from the transient to steady is decided and enforced by the sink based on the obtained sensor data which enables the sink to identify a steady threshold interval and initiate the necessary superframe change. The identified threshold interval is communicated to all the sensor nodes which update their local information on the threshold interval. The mode-switch from steady to transient initiated by the sensor nodes is a critical mode-switch operation as a failure to switch to transient mode will effect the control of the process.

The switch from steady to transient can be affected by packet failures in two ways: (1) The alert packets could be lost resulting in a delay in alerting the sink of the threshold breach; and (2) the notification packet requesting the entire network to switch operational mode could fail to reach certain nodes, resulting in some nodes continuing in the steady mode operation. Concerning (2), then the protocol has been designed such that the steady superframe is a multiple of the transient superframe. This means that the protocol can tolerate a delayed switch of operation model for some nodes as it allows the nodes to later synchronise to the operational mode in a case where they miss notification messages for a mode-switch due to packet loss. Also, within each notification message, the sink includes the information of the mode the network is currently operating in. In case the node is not in the same mode as the sink, it can synchronise immediately when such a message is received.

3. TinyOS and nesC Protocol Implementation

TinyOS is an operating system designed for low-power embedded systems devices as used in WSANs. TinyOS applications and protocols are implemented in nesC[10] (a dialect of the C programming language). The architecture of an application in TinyOS is organised as a graph consisting of *components* connected via *interfaces*. The *components* are entities that provide and use *interfaces*. An *interface* is typically used to define a type of service and the signature of a component lists the interfaces it uses and provides. Below we discuss the implementation model including integration of the DMAMAC protocol into the TinyOS platform. The DMAMAC protocol mainly require interfaces connecting to the low-level hardware and radio communication, a time-synchronisation service, and an application to demonstrate the use of the DMAMAC protocol. In addition, we use interfaces that facilitate receiving packets from the higher levels, i.e., application and network.

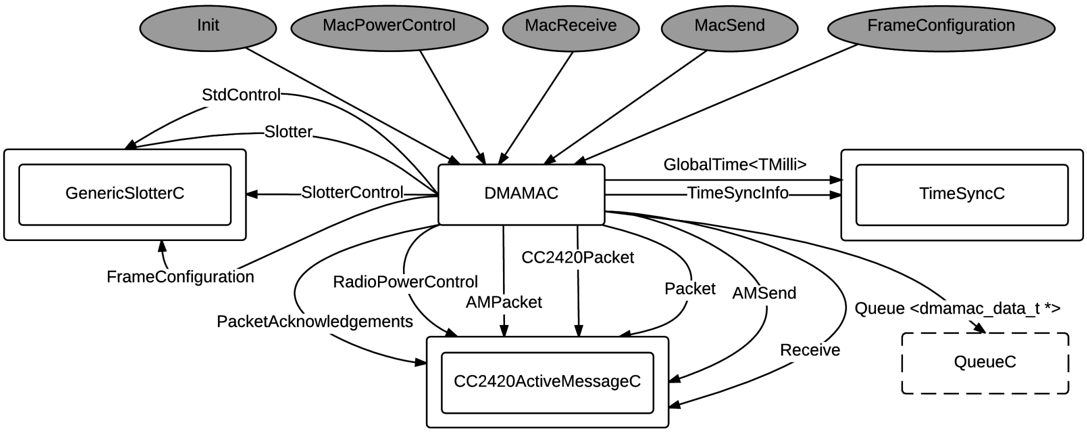


Fig. 1: Component graph for the DMAMAC protocol implementation

Component software architecture. The architecture component graph for the DMAMAC nesC implementation is shown in Fig. 1. A module is indicated by a box with a single line, and boxes with a double line specifies configuration of a component. The box with a dashed line denotes a generic module. The grey ovals are interfaces being provided. Generic modules of components similar to Timers can be instantiated multiple times as per application requirement. We rely on several existing components provided by TinyOS 2.1.2 and re-use the MAC Layer Architecture[6] services including Frame Configuration and Slotter Control in the implementation of the DMAMAC protocol. These services are provided by the GenericSlotterC component and are part of the TinyOS distribution (contributions).

Interfaces. The DMAMAC component (Fig. 1) *uses* several components via their provided interfaces to implement the functions as per the protocol specification. We re-use most of the components rather than implementing new ones. The most important components that are used via their provided interfaces are:

- GenericSlotterC providing slotting functionality for the TDMA in steady and transient super frames.
- CC2420ActiveMessageC is the radio component and provides all the radio packets related interfaces.
- TimeSyncC providing the time synchronisation required to keep all the nodes in the network synchronized.
- QueueC for packet buffer queues to store incoming and outgoing packets until handled.

The Frame Configuration service from the GenericSlotterC component is used to define the parameters of the transient and steady superframe. The parameters defined include *slotsize* and *superframe length*. We use a *slotsize* of 10 ms, a superframe length of 100 slots for transient mode and 200 slots for steady mode. The Frame Configuration is also used to switch superframe's dynamically between steady and transient superframes. For TDMA slot scheduling and control, the Slotter Control service is used. For packet handling we use the CC2420ActiveMessageC interfaces provided by the radio module of the Z1 hardware. The packet sending and receiving interfaces are AMSend and Receive, respectively. The PacketAcknowledgements interface is used for obtaining acknowledgements for transmitted packets and the AMPacket interface is used to for packet data access. The CC2420Packet component provides link data including Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI) used for the analysis (see Sect. 4). RadioPowerControl is an interface used to switch radio on or off.

The interfaces *provided* by the DMAMAC component are: Send, Receive, MacPowerControl, Init and FrameConfiguration. The Send and Receive interfaces are provided for packet handling in both directions: incoming and outgoing. The *Init* interface is used for initialising the module, after which MacPowerControl is used for switching the component on and off. The *FrameConfiguration* is provided for modifying the transient and steady superframe length if required by the application.

Time Synchronisation. Time synchronisation is a key element of any TDMA-based MAC protocol as all nodes must start start on every slot simultaneously. Given the clock drift and varying local time on each nodes, time synchronisation is necessary. We use the Flooding Time Synchronisation Protocol (FTSP)[11] provided with the TinyOS suite to obtain time synchronisation for DMAMAC. A root node is selected (the sink in our case) and the time of the root node is used as global time by the rest of the nodes. The root node sends out a time-synchronisation message at a user defined interval and all nodes update their local clock based on the values received from the root node. The TimeSyncC component provides the time synchronisation required and the the timing information of each node can be obtained via the TimeSyncInfo interface. Alternatively, we could have used the CC2420TimeSyncMessage component to send time synchronisation information in the notification slot of the sink. But, this would require integrating the time synchronisation implementation into our MAC level implementation. In order to obtain a modular design and demonstrate that DMAMAC can be implemented with a standard time synchronisation component, we directly use the FTSP time synchronisation as a background process. We use a synchronisation rate of 3s, i.e., a time synchronisation message is sent from the root node to all nodes every 3s.

Mode-switch implementation. The FrameConfiguration interface is used to implement the mode-switch which can occur in two ways: (1) Transient to steady which is a non-critical switch implemented by the sink based on the readings of the sensors and signalled via notification to all the nodes; and (2) Steady to transient which is a safety-critical switch decided by the sink based on the alert messages transmitted by one or more sensor.

When a node receives a notification message from the sink about change of mode, it uses the FrameConfiguration interface to change the type of superframe used. The loss of a notification packet can affect the mode-switch procedure and thus the process control adversely. This is mainly important for the critical steady-to-transient switch. We therefore implement the notification messages from the sink such that they repeat the current mode in every round. The design of the superframes is such that steady mode superframe is a multiple of the transient mode superframe which means that even in a case where some nodes are not in the same mode (due to notification loss), they can easily synchronise in the next notification message. In addition, the reliability of the protocol can be further enhanced by ensuring that if a node detects that it misses a notification slot, then it switches into transient mode as a safety precaution. In case where the network is still in steady mode, the node will re-synchronise back to steady mode as soon as it correctly receives payload in a notification slot. In the implemented version of the DMAMAC protocol, we included an additional notification slot in the beginning of the sleep parts of transient length (Nt) in the steady superframe as shown in Fig. 2. This is a modification to the original superframe structure[9] to improve robustness and reduced switch delay.

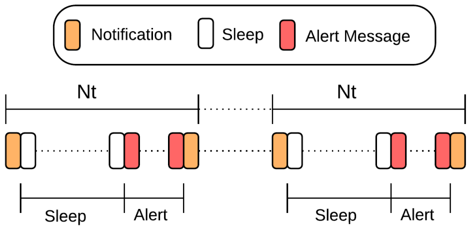


Fig. 2: Modified sleep part in the steady superframe

4. Deployment and Experimental Evaluation

The zolertia nodes used for the experiment have a MSP430F2617 low power micro-controller with a 16-bit 16-MHz RISC CPU, 8KB RAM and 92KB of flash memory, and a CC2420 IEEE 802.15.4 complaint transceiver operating at 2.4GHz with a data rate of 250Kbps. The multi-hop network topology used for the experiments is shown in Fig. 3(left). It consists of 3 sensor nodes, 1 actuator node, and a sink node. The senors relay sensor information to the sink and receive notification and data packets (to forward) from the sink. The actuator node only receives data and notification packets. The configuration of the testing application which uses the DMAMAC protocol for communication in the experiments is shown in Fig 3(right). The application mainly uses the DMAMAC protocol to send and receive packets, and also to modify frame configuration (i.e., length of steady and transient) as required. The application uses the generic module TimerMilliC to produce packets at a preset interval. It also initialises (boots) the TimeSyncC component for use by the DMAMAC component.

The DMAMAC protocol has been designed for process control application which means that it will typically be deployed in a factory automation setting where there would be numerous machines present creating noise that would effect the operation of the protocol. The noise could be a result of the variation in temperature, vibrations, and electromagnetic noise[12]. To emulate a real process automation environment, we have used an electrical laboratory containing multiple running motors. This allows us to setup a controlled environment in which we can conduct

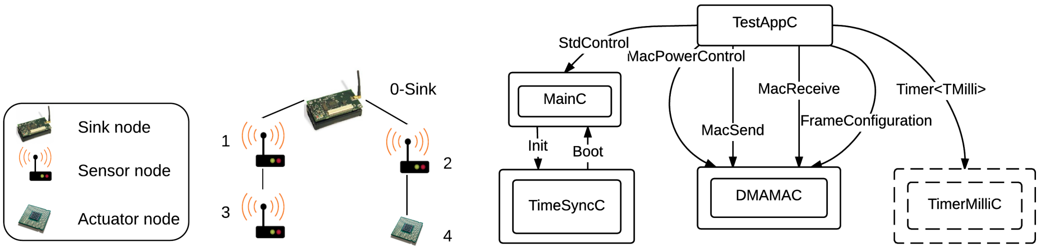


Fig. 3: Experimental setup: network topology (left) and application structure (right)

experiments in different scenarios to study the effect of external factors like motor noise on the performance of the protocol.

Our first goal was to validate that the implementation correctly sets the radio states. For this, we measured the current to validate the change of radio states (TX, RX and SLEEP). The values obtained are shown in Tab. 1. The measurements were made using an oscilloscope with a 3V power source placed with node and a 10 Ω resistor coupled in series. The measured current are in the expected range as per the data sheets for the radio hardware.

Radio state	Current measured
Sleep	0.5 mA – 0.6 mA
TX	21 mA – 22 mA
RX	23 mA – 23.5 mA

Table 1: Measured current in radio states.

Our second goal is to study the impact of noise on packet failures and how it affects the mode switch procedure of the protocol. We consider two separate scenarios: a motor room scenario and open room scenario. We measure three metrics: Received Signal Strength Indicator (RSSI), Link Quality Indicator (LQI), and failure of notification packets. We mainly focus on notification packets and parameters measuring its quality because they are directly responsible for the network wide mode-switch. For the open room scenario, we perform two experiments with varying setup, one with line of sight placement and one non-line of sight or arbitrary placement (both with 5 nodes) of nodes. For the motor room scenario we used the same setup, but with additional running motors. The two scenarios are further discussed below. RSSI values range from -50 dBm to -95 dBm (maximum receiver sensitivity), where values towards -50 dBm are considered to be better. RSSI is the measurement of the power in the received signal and thus gives some indication of pathloss for the signal. LQI values range from 50 to 110 for the CC2420, where values towards 110 are the good ones. LQI is calculated for each packet in TinyOS and indicates link quality based on packet reception.

Scenario 1: Motor room. These experiments were conducted in a laboratory with multiple asynchronous induction motors with high wattage (3kW and generating a voltage of 325 V). The idea is to test DMAMAC in an industry-like scenario where motors are running and generating noise including electromagnetic noise. This scenario also includes frequency controllers, and current converters (Direct to Alternating) operating at varying frequencies. During the experiment, we also had students that walked around these motors performing tests. The laboratory in addition includes standard Wi-Fi access points. We considered two placement of the nodes: line of sight placement and (arbitrary) non line of sight placement. The line of sight placement has the nodes places such that they are visible to each other. The arbitrary placement is around the motors with no direct line of sight.

The graphs from the obtained RSSI and LQI measurements from the line of sight placement scenario experiment are shown in Fig. 4(left) and Fig. 4(right). Rounds define the number of notifications received. It can be seen that the link quality and the received signal strength are quite stable. But in a general industrial environmental setup, line of sight placement is not always possible. This is why we also also consider node placement in order to get a more accurate picture of the performance of the hardware. The graphs for the obtained RSSI and LQI measurements from the non line of sight (arbitrary) placement scenario are shown in Fig. 5(left) and Fig. 5(right). Observing these figures, we can see that the link quality is relatively stable, but the RSSI has a large variation through the experiment. Given the collective picture of LQI and RSSI and packet loss, it can be concluded that the arbitrary placement does affect the link quality for the used hardware based on the obstacles and the generated noise.

Scenario 2: Open room. The open room scenario uses the same laboratory as the motor room scenario but without any running motors or students around, but Wi-Fi was active in both scenarios. The RSSI values obtained in the open

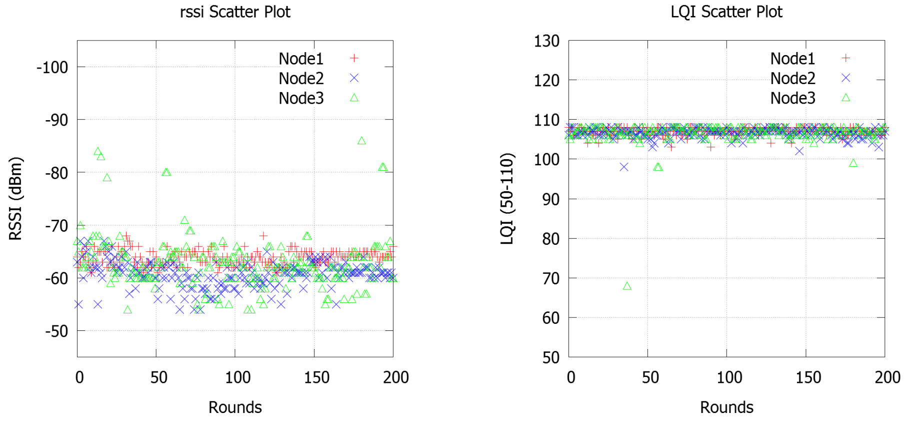


Fig. 4: Experimental results for motor room scenario - line of sight placement.

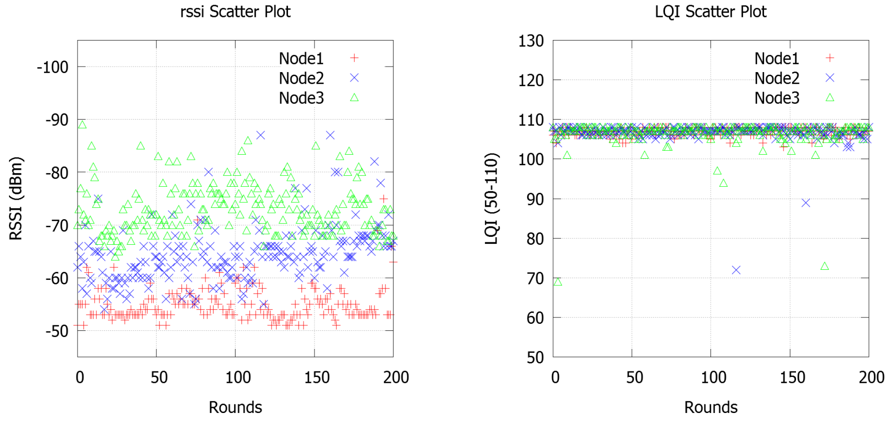


Fig. 5: Experimental results for motor room scenario - arbitrary node placement.

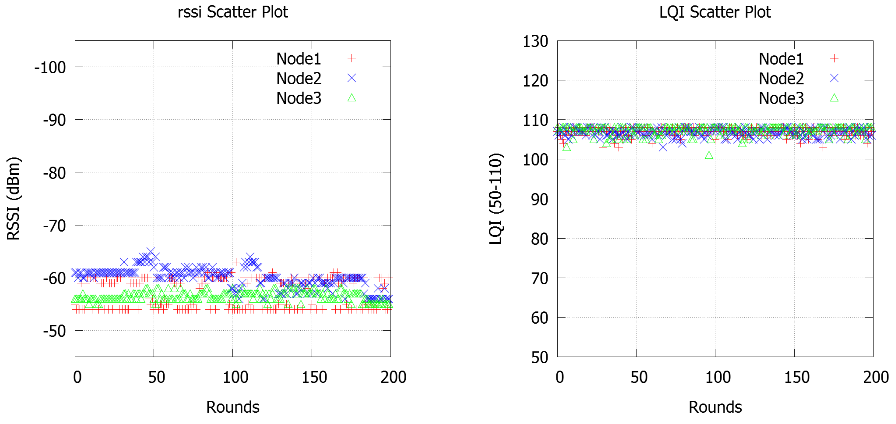


Fig. 6: Experimental results for the open room scenario.

room scenario is shown in 6(left) and the LQI is shown in 6(right). In this scenario, we investigated both line of sight and (arbitrary) non-line of sight placements. The results obtained for both placements are fairly similar and we therefore only include the line of sight experiment due to space limitation. It can be seen that in comparison with the motor room scenario, the RSSI values are fairly stable over the time, and so are also the the LQI values.

Scenario	Open room non-LOS	Open room LOS	Motor room non-LOS	Motor room LOS
Node 1	0/300	0/300	1/300	3/300
Node 2	1/300	1/300	3/300	3/300
Node 3	6/300	2/300	6/300	4/300

Table 2: Notification packet failures on nodes 1-3 line of sight (LOS) and non-line of sight (non-LOS).

The results obtained in both the motor room and the open room scenario show two main things. The notification packet failure is small and is similar across the scenarios and experiments as summarised in Table 2. Thus, the protocol runs across these scenarios with little packet failure. The LQI values shows minimum variation in link quality in all experiments. Primarily, we can infer that the environment conditions effects the performance of the protocol, and that the steps taken to make the protocol robust indeed help in these situations. The RSSI measure on the motor room scenario shows large variations which could be an effect of multi-path fading in the non-line of sight scenario. Also, in the line of sight scenario the noise could be a factor. Continuously operating Wi-Fi in the area could also create interference, but has similar effect on both scenarios. However, additional experiments are needed to get definite conclusions on the multi-path effects.

5. Conclusions and Future Work

We have demonstrated the practical implementability of the dual-mode DMAMAC protocol for process control. In particular, we have shown how to implement the mode-switch which is a distinct feature of DMAMAC not found in other MAC protocols for process automation. Also, we have shown that DMAMAC can be implemented based on a standard time synchronisation protocol. Our implementation work has also resulted in two improvements to the robustness of the DMAMAC protocol in tolerating packet loss. We have evaluated the DMAMAC implementation in a representative process control automation environment which included sources of noise that interfered with the radio communication and hence affect the protocol operation. These results demonstrated the capability of the DMAMAC protocol to operate in a satisfactory manner in the presence of packet failures, in particular that the critical steady to transient mode switch was performed properly.

The result presented in this paper combined with our earlier work[9] on design and simulation-based evaluation of DMAMAC can be seen as the final step in the development of DMAMAC. A natural extension of the present work would be a large-scale deployment testing of the protocol in order to investigate scalability. Investigating the use of DMAMAC in less safety-critical contexts such as home automation is also a possible direction of future work.

References

1. I. F. Akyildiz, I. H. Kasimoglu, Wireless Sensor and Actor Networks: Research challenges, Ad Hoc Networks 2 (4) (2004) 351–367.
2. A. A. Kumar S., K. Øvsthus, L. M. Kristensen, Towards a Dual-Mode Adaptive Mac Protocol (DMA-MAC) for feedback-based Networked Control Systems, in: The 2nd International Workshop on Communications and Sensor Networks, 2014, pp. 138–162.
3. P. Suriyachai, J. Brown, U. Roedig, Time-critical data delivery in Wireless Sensor Networks, in: DCOSS, Vol. 6131, 2010, pp. 216–229.
4. P. Suriyachai, U. Roedig, A. Scott, Implementation of a MAC protocol for QoS support in Wireless Sensor Networks, in: IEEE International Conference on Pervasive Computing and Communications, 2009, pp. 1–6.
5. J.-H. Hauer, TKN15. 4: An IEEE 802.15. 4 MAC implementation for TinyOS.
6. K. Klues, G. Hackmann, O. Chipara, C. Lu, A component-based architecture for power-efficient media access control in wireless sensor networks, in: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, SenSys 07, 2007, pp. 59–72.
7. Z1 datasheet, zolertia, `http://www.zolertia.io//`, accessed: 25-01-2016.
8. P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler, Tinyos: An operating system for sensor networks, in: W. Weber, J. Rabaey, E. Aarts (Eds.), Ambient Intelligence, Springer, 2005, pp. 115–148.
9. A. A. Kumar Somappa, L. M. Kristensen, K. Ovsthus, Simulation-based evaluation of DMAMAC: A dual-mode adaptive MAC protocol for process control, in: SIMUTools, 2015, pp. 218–227.
10. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, The nesc language: A holistic approach to networked embedded systems, in: Proceedings of the Conference on Programming Language Design and Implementation, PLDI ’03, 2003, pp. 1–11.
11. M. Maróti, B. Kusy, G. Simon, A. Lédeczi, The flooding time synchronization protocol, in: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys ’04, 2004, pp. 39–49.
12. K. S. Low, W. Win, M. J. Er, Wireless sensor networks for industrial environments, in: International Conference on Computational Intelligence for Modelling, Control and Automation, Vol. 2, 2005, pp. 271–276.

Chapter 11

Towards a Model-Based Development Approach for Wireless Sensor-Actuator Network Protocols

Towards a Model-Based Development Approach for Wireless Sensor-Actuator Network Protocols

Position paper

A. Ajith Kumar S.
Bergen University College, Norway
University of Agder, Norway
aaks@hib.no

Kent I. F. Simonsen
Bergen University College, Norway
Danish Technical University, Denmark
kifs@hib.no

ABSTRACT

Model-Driven Software Engineering (MDSE) is a promising approach for the development of applications, and has been well adopted in the embedded applications domain in recent years. Wireless Sensor Actuator Networks consisting of resource constrained hardware and platform-specific operating system is one application area where the advantages of MDSE can be exploited. Code-generation is an integral part of MDSE, and using a multi-platform code generator as a part of the approach has several advantages. Due to the automated code-generation, it is possible to obtain time reduction and prevent errors induced due to manual translations. With the use of formal semantics in the modeling approach, we can further ensure the correctness of the source model by means of verification. Also, with the use of network simulators and formal modeling tools, we obtain a verified and validated model to be used as a basis for code-generation. The aim is to build protocols with shorter design to implementation time and efforts, along with higher confidence in the protocol designed.

Keywords

Model-Driven Software Engineering (MDSE), Wireless Sensor-Actuator Networks (WSAN), Code Generation, Validation, Verification, Simulation, Embedded Software

1. INTRODUCTION

Wireless Sensor-Actuator Networks (WSANs) consists of a network-connected sensors and actuators working towards a specific mission. It is a branch of the existing Wireless Sensor Network (WSN) systems. Sensors and actuators in WSAN are resource constrained small devices usually powered by batteries. WSAN has several application domains such as process automation and factory automation, and is thus widely applicable in an industrial setting. One important setting in which WSAN is applicable is the control-loop of automation processes. Systems with control-loops have stringent requirements, and these applications are often safety-critical. This implies that it is important to have a sound design methodology for developing software solutions to be deployed on the sensors and actuators. Quite often the design methodology includes modeling of the protocols. Later these models are manually converted to simulation code and further analyzed. Some development approaches also includes model-checking to check for correctness. As the last step, models are transformed to the implementation platform code. This approach towards design of WSAN solutions can be combined with existing software engineering approaches, to strengthen the reliability of the software generated and to reduce the time for development.

Model-Driven Software Engineering (MDSE) is one such approach that has long been seen as a prominent approach for software engineering. MDSE uses models as primary artifacts. MDSE is an extensively used methodology across several domains for development of applications. Advantages of MDSE approach include shorter time from design to implementation, verified and validated models used for automatic code generation. In an industrial setting, continuous work is done in reducing the cost of developing software and the time required, and MDSE works towards this. In MDSE approaches, the initial model is an abstract and platform independent representation of the protocol. This abstraction allows the designers to focus on creating a model with proper functioning. Combining this abstraction step with formal approaches allows further improving the verification and validation process. The abstract models can be model-checked for verification and can be further simulated to obtain initial performance assessment results. This minimizes the possibility of errors in the code generated to a further extent.

One tool that allows model-checking and simulation is Coloured Petri Nets (CPN) Tools [5]. It is based on the expressive language CPN combined with the Standard ML programming language. CPN has been previously used for modeling and verification of network protocols [1]. This combined with the PetriCode [14] tool, forms a complete MDSE approach for development of network protocols and can be applied to develop WSAN protocols as well. PetriCode is a code generation tool that takes platform-independent CPN models as input and produces platform-specific code for various platforms like Java, Clojure and Groovy. The aim of this work is to extend PetriCode to support platform-specific code generation for sensor network

CyPhy '14, April 14-17 2014, Berlin, Germany
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2871-5/14/04 ...$15.00.
http://dx.doi.org/10.1145/2593458.2593465

platform such as the TinyOS [9] and for sensor network simulators. Using a CPN model as a starting point allows us to base our implementations on verified and functionally correct model, giving confidence in the correctness of the generated code. In this article, we mainly focus on providing an MDSE approach for protocol development for WSAN using CPN models and the PetriCode tool. Generation for disparate platforms (MiXiM [6],TinyOS) is still challenging because the model must then support several programming models in the implementations. For the model, however this can be seen as an advantage as it forces the model to focus on the logical operation of the protocol and not include implementations details. As a case study we use the GinMAC [15] protocol specification. GinMAC protocol was designed for WSAN, with strict packet delay requirements.

2. RELATED WORK

Model-driven software engineering has been used in the WSN domain for a while now [10, 18]. There have been several work proposing various frameworks for rapid development of WSN protocols, dominantly based on Doman Specific Modeling Languages (DSML) [2, 4] or the Unified Modeling Language (UML) [10, 18, 13]. In [11], a design framework is proposed which facilitates behavior simulation, and multi-platform code-generation. It requires multiple steps for platform-specific code generation and the simulation done is very basic. In [12], an architectural framework, Architecture for Wireless Sensor and Actuator Network (ArchWiSeN) is proposed. This architecture is based on UML diagrams as platform independent high level models and consider TinyOS platform for code generation and simulation was performed on TOSSIM. Moppet [2] is an MDSE based method that uses feature modeling, in-tool performance estimator and code generation for TinyOS. In [4] the authors concentrated mainly on the modeling aspects proposing an architecture consisting of separate modeling languages for environment, software architecture and node modeling. Mapping is used to create relation between these modeling languages. They also propose code generation as possible future work based on existing model to text generators. In [16], a model-driven development approach is proposed based on a Domain Specific Language (DSL) for WSN. They perform a model-to-model transformation from the initial platform independent model to a platform specific model. This platform specific model (TinyOS) is then used for code generation. In [17] a code-generation technique for nesC was provided. The article [17], focussed only on the code-generation part and specifically the TinyOS platform, thus creating platform-specific code generation software. The key differences between the existing and our MDSE approach are: firstly we provide a formal platform to begin modeling with, which also provides simulation and state-space analysis possibilities, secondly we provide conversion possibility to event-based network simulator (MiXiM [6]) that has been used in WSN research. These features are provided along with code-generation for a sensor specific platform like TinyOS.

3. OVERVIEW OF APPROACH

In this section, we provide an overview of our proposed MDSE approach for protocol design and implementation. The approach is illustrated in figure 1. The starting point of the approach is an abstract model of the given protocol. For modeling purposes, we use CPN Tools [5]. Along with modeling and verification, CPN Tools allows for initial simulation. Using CPN Tools, developers can create a formal executable model of a protocol. Developers can also perform behavioral and functional verification, state-space analysis, and initial simulation. Based on the analysis, developers can verify and validate the given model, based on its requirements specification. In the next step, the refined CPN model is converted to code models using the PetriCode tool [14]. PetriCode is a code generation tool that is designed to automatically generate implementations of network protocols for various platforms. PetriCode requires the models to be annotated with *code generation pragmatics*. These pragmatics are structural annotations on CPN models that are bound to *code-generation templates* via *template bindings*. In PetriCode the pragmatics and the template bindings as well as the structure of the CPN model is used to guide code generation without needing to translate or interpret the ML code of any given CPN model. Using the PetriCode tool, developers can automatically generate code for model-checkers, network simulators, and hardware platforms. The implementation generated for the simulation and model checking platforms are used to perform further analysis. This can further strengthen the developer's confidence in the correctness of the final implementation.

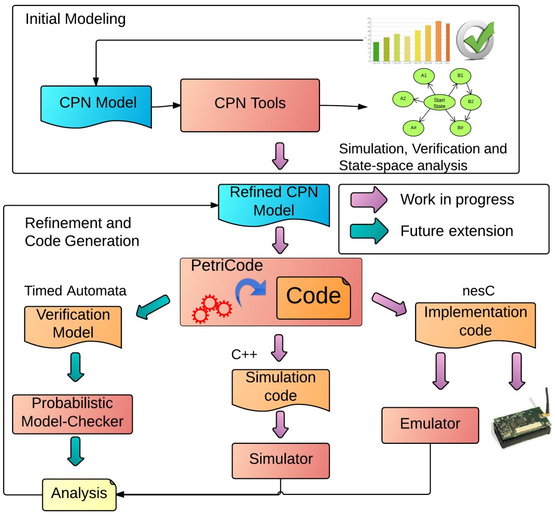


Figure 1: Model-based Development Approach

For the initial work, we have selected certain tools namely, Uppaal [3] /PRISM [7] for model-checking, Omnet-Mixim [6] for network simulation, and TinyOS [9] for platform-specific simulation and implementation. All these have been successfully used in a number of application case studies. Using this approach, before the final implementation, the model can be further tested using the tools listed above. Uppaal/PRISM are powerful probabilistic model-checkers that allow for further exhaustive behavioral and functional analysis. Given the stochastic nature of the wireless channel used for communication in WSAN, it is essential to perform a probabilistic study for the model-checking procedure. Omnet-Mixim is a discrete event simulator that allows for network simulations using pre-defined wireless channel models and also has rich protocol library thus provides necessary infrastructure. From the results obtained by

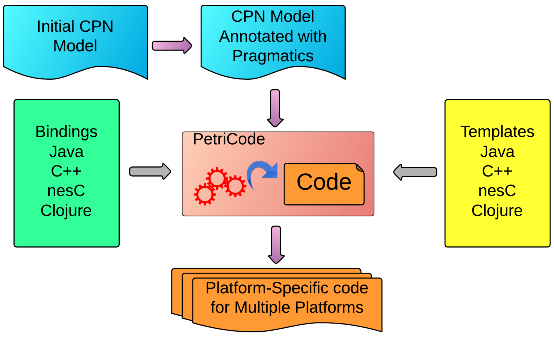

Initial CPN Model
CPN Model Annotated with Pragmatics
Bindings Java C++ nesC Clojure
PetriCode
Code
Templates Java C++ nesC Clojure
Platform-Specific code for Multiple Platforms

Figure 2: Code generation architecture

model-checking and simulating the model, the CPN model can be further refined to eliminate possible design flaws and to provide high quality software for the implementation. The event based sensor network platform TinyOS is used for hardware implementation code generation. TOSSIM [8] is the emulator for TinyOS platform, thus nesC code can be simulated.

The flexibility of PetriCode tool that makes it possible to generate code for different platforms is an important advantage over most other existing tools. Developers can create their own templates and bindings for a platform based on the pragmatics defined by the tool. For the design, analysis, and implementation of protocols, platform specific codes in C++ for simulation, Timed Automata for model checking, nesC for TinyOS and TOSSIM is required. The challenge in the current work is to be able to extract multi-platform representations out of a single CPN model. Given that different platforms such as TinyOS which has a event-based code structure, Java and C++ have object-oriented structure, it is essential to exploit the similarity in them to be able to generate code from a single model.

4. MODELS AND CODE GENERATION

An important part of MDSE is code-generation. It is essential towards reducing the errors in the coding process and reducing time required to generate code from design. In our framework, we use PetriCode tool [14] to be able to generate platform-specific code and simulation tool code. Our aim is to extend the PetriCode tool to provide implementation and simulation code. The code generation framework is shown in figure 2. The code generation is carried out in four steps, detailed below.

1. Create a CPN model of the protocol for which an implementation is to be obtained.
2. Annotate modeling elements with pragmatics to facilitate code generation. PetriCode models have an explicit control-flow path that is defined by <<Id>> pragmatics. This allows PetriCode to use the CPN model structure to generate code for programming languages of several paradigms
3. Create bindings and templates for the platform to which the resulting code has to be generated.
4. Use PetriCode to generate code using proper bindings.

Multiple steps are involved in going from CPN model to a platform-specific code for which [14] can be referred. In this article, we use PetriCode tool and extend its functionalities to support WSAN protocol development. The bindings and the templates required for the code-generation for every platform has to be specified. These templates and bindings use the pragmatics to interpret the CPN model and create the corresponding implementation code. The PetriCode tool already has predefined bindings for Java, Clojure and Groovy. A sample CPN model for a sensor looks like the one in figure 3. This is the top most level of the layered CPN model, defining a sensor node and is annotated with pragmatics <<Principal()>> and <<Channel()>>. The top most level depicts the layered sensor model with a radio component at the lowest layer (OSI layering), connecting the sensor with the communication channel. Above it, is the Medium Access Control (MAC) layer which handles all the communication through and to the sensor. The application and network part are combined in this model which represents application support and routing decision support protocols. A level lower where the module is defined in detail, inside the Medium Access Control (MAC) module we can see <<Service()>> pragmatic as shown in figure 4. The figure depicts two services being handled in the MAC module: radio packet handler and upper layer packet handler. The module also consists of a variable declaration *SingleSlotBuffer* used by the upper layer packet handler service.

Similar to relating programming concepts between different platforms it is a challenge to relate pragmatics to different code structures across languages. A view of the differences in concepts between the initial pragmatics annotations and platform-specific languages is shown in figure 5. The nesC code structure consists of components, configurations, modules and interfaces. Whereas C++ consists of objects, classes, methods and abstract classes. Initially, we map <<Principal()>> pragmatic to objects in C++ and components in nesC. At the lower level, nesC contains of commands and events (synchronous and asynchronous) and C++ consists of methods. We map the <<Service()>> pragmatic to commands and events in nesC, and to methods in C++. PetriCode already consists of Java templates and bindings. We need to create new bindings and templates for C++ and nesC, adhering to their implementation rules. An example C++ code generated from the CPN model defined in figure 3 is shown in listing 1. For this generation, the binding used is shown in listing 2 and the template example for "Declaration" is shown in listing 3. An example nesC code (skeleton) generated for the same CPN model is shown in listing

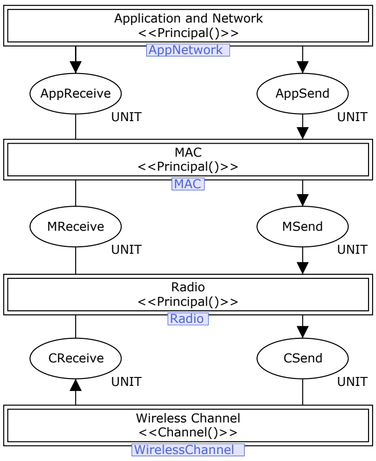


Figure 3: CPN model of a Sensor

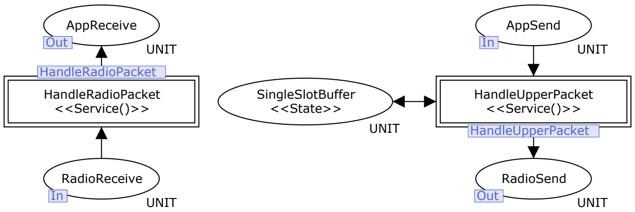


Figure 4: CPN model of the Medium Access Layer

4. The "Declaration" binding is used to generate code for the pragmatic <<State>> which can be seen in figure 4. To obtain the nesC code, we plan to use asynchronous commands and events. An important constituent of a program code block is its control flow. Given the CPN's lack of enforcing a structure particularly to enforce control flow, PetriCode defines certain pragmatics and control flow structure for the <<Service()>> level [14]. Thus at the service level, we need to follow a particular structure that defines the control-flow while transforming the model from an initial CPN model to an annotated PetriCode CPN model. In the current work, we are in the process of creating code-generation for simulation and hardware platforms. Currently an abstract model of the protocol has been created and a C++ conversion (skeleton code) is generated as an example.

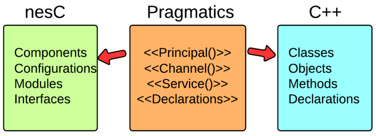


Figure 5: Concept mapping from Pragmatics to nesC and C++

Listing 1: C++ source code file

```
//Mac c++ file
#include <BaseMacLayer.cc>
#include "MAC.h"

Object SingleSlotBuffer;
void MAC::HandleUpperPacket() {
    /*[]*/ /*[]*/
    /*vars: [__TOKEN__:]*/
    Object __TOKEN__ = null;
}
void MAC::HandleRadioPacket() {
    /*[]*/ /*[]*/
    /*vars: [__TOKEN__:]*/
    Object __TOKEN__ = null;
}
```

Listing 2: Template Bindings file

```
//C++.bindings
principal(pragmatic: 'principal',
    template: "./cTmpl/mainClass.tmpl")
service(pragmatic: 'service',
    template: "./cTmpl/externalMethod.tmpl")
DECLARATIONS(pragmatic: '_-DECLARATIONS-_',
    template: './cTmpl/__DECLARATIONS__.tmpl')
```

Listing 3: Template file for declarations binding

```
//__DECLARATIONS__.tmpl
/*vars: ${vars}*/
<%vars.each{%>
<%if( it != '[]' && it != 'msg'){%>Object ${it} = null;
<%}}%>
```

Listing 4: nesC implementation code

```
//MAC Component
module MAC{}
implementation {
Object SingleSlotBuffer;
event void HandleUpperPacket() {
    /*[]*/ /*[]*/ /*vars: [__TOKEN__]*/
    Object __TOKEN__ = null;
}
event void HandleRadioPacket() {
    /*[]*/ /*[]*/ /*vars: [__TOKEN__]*/
    Object __TOKEN__ = null;
    }
}
```

4.1 Current Challenges

The important challenges in the current project using PetriCode for code-generation for multiple platforms are listed below:

1. Finding proper abstractions and abstraction level for the platform-independent models.
2. PetriCode assumes a certain control-flow model in the models. Even though PetriCode has been used to generate code for languages representing different programming paradigms, it may not be entirely trivial to adapt to target platforms and simulators for the embedded domain.
3. Making code-generation as complete as possible with regards to being able to generate most or any protocol with little or no need to create new pragmatics and templates for various target platforms.

5. OUTLOOK

In this article, we have proposed an MDSE approach for designing protocols for WSN and WSAN applications. We carry out high-level abstract modeling in feature rich CPN Tools which also allows for initial verification and simulation unlike other existing tools. We further provide the opportunity to also perform simulation on full-fledged network simulators that focus specifically on network simulations and have been developed over the years to incorporate various wireless features. Thus, initially we extend PetriCode to provide code generation for the network simulator MiXiM and sensor network platform TinyOS in nesC language. The extensions have been partly completed and the work is in progress. We have generated initial skeleton code for C++ and nesC. Given the model of the PetriCode tool, it can also be extended towards providing conversions to other platforms as well as model-checking tools for further verification, essentially probabilistic verification considering the complex nature of wireless channels involved. We can also easily extend to obtain possible code generation to other prominent sensor network platforms namely ContikiOS as well, which is based on the C language. As a case study, we plan to design the GinMAC protocol according to its specifications and generate implementation code.

6. REFERENCES

[1] J. Billington, G. Gallasch, and B. Han. A coloured petri net approach to protocol verification. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 210–290. Springer Berlin Heidelberg, 2004.

[2] P. Boonma and J. Suzuki. Model-driven performance engineering for wireless sensor networks with feature modeling and event calculus. In *Proceedings of the 3rd Workshop on Biologically inspired Algorithms for*

Distributed Systems (BADS), pages 17–24. ACM, 2011.

[3] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. Vliet, and Z. Wang. Statistical model checking for networks of priced timed automata. In *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 6919 of *LNCS*, pages 80–96. Springer Berlin Heidelberg, 2011.

[4] K. Doddapaneni, E. Ever, O. Gemikonakli, I. Malavolta, L. Mostarda, and H. Muccini. A model-driven engineering framework for architecting and analysing wireless sensor networks. In *Proceedings of the 3rd International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 1–7, 2012.

[5] K. Jensen, L. Kristensen, and L. Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9:213–254, 2007.

[6] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K. Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in omnet++ the mixim vision. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops (Simutools)*, number 71, pages 1–8, 2008.

[7] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer Berlin Heidelberg, 2011.

[8] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded networked sensor systems (ACM SenSys)*, pages 126–137. ACM, 2003.

[9] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg, 2005.

[10] F. Losilla, C. V Chicote, B. Alvarez, A. Iborra, and P. Sanchez. Wireless sensor network application development: An architecture-centric MDE approach. In *Software Architecture*, Lecture Notes in Computer Science, pages 179–194. Springer Berlin Heidelberg, 2007.

[11] M. M. R. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri. A framework for modeling, simulation and automatic code generation of sensor network application. In *Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 515–522, 2008.

[12] T. Rodrigues, T. Batista, F. Delicato, P. Pires, and A. Zomaya. Model-driven approach for building efficient wireless sensor and actuator network applications. In *Proceedings of the 4th International Workshop on Software Engineering for Sensor Network Applications(SESENA)*, pages 43–48, 2013.

[13] R. Shimizu, K. Tei, Y. Fukazawa, and S. Honiden. Model driven development for rapid prototyping and optimization of wireless sensor network applications. In *Proceedings of the 2nd International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 31–36, 2011.

[14] K. I. F. Simonsen, L. Kristensen, and E. Kindler. Generating protocol software from cpn models annotated with pragmatics. In *Formal Methods: Foundations and Applications*, volume 8195 of *Lecture Notes in Computer Science*, pages 227–242. Springer Berlin Heidelberg, 2013.

[15] P. Suriyachai, J. Brown, and U. Roedig. Time-critical data delivery in wireless sensor networks. In *Proceedings of Distributed Computing in Sensor Systems (DCOSS)*, pages 216–229. Springer Berlin Heidelberg, 2010.

[16] N. X. Thang, M. Zapf, and K. Geihs. Model driven development for data-centric sensor network applications. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '11, pages 194–197. ACM, 2011.

[17] V. Veiset and L. M. Kristensen. Transforming platform independent cpn models into code for the tinyos platform: A case study of the RPL protocol. In *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE)*, volume 989, pages 259–260. CEUR workshop proceedings, June 2013.

[18] C. Vicente-Chicote, F. Losilla, B. Alvarez, A. Iborra, and P. Sanchez. Applying MDE to the development of flexible and reusable wireless sensor networks. *International Journal of Cooperative Information Systems*, 16:393–412, 2007.

Chapter 12

Model-based Development for MAC Protocols in Industrial Wireless Sensor Networks

Model-based Development for MAC Protocols in Industrial Wireless Sensor Networks

Admar Ajith Kumar Somappa[1,2] and Kent Inge Fagerland Simonsen[1]

[1] Bergen University College, Bergen, Norway
[2] University of Agder, Grimstad, Norway
{aaks,kifs}hib.no

Abstract. Model-Driven Software Engineering (MDSE) is an approach for design and implementation of software applications, that can be applied across multiple domains. The advantages include rapid prototyping and implementation, along with reduction in errors induced by humans in the process, via automation. Wireless Sensor Actuator Networks (WSANs) rely on resource-constrained hardware and have platform-specific implementations. Medium Access Control (MAC) protocols in particular are mainly responsible for radio communication, the biggest consumer of energy, and are also responsible for Quality of Service (QoS). The design and development of protocols for WSAN could benefit from the use of MDSE. In this article, we use Coloured Petri Nets (CPN) for platform independent modeling of protocols, initial verification, and simulation. The PetriCode tool is used to generate platform-specific implementations for multiple platforms, including MiXiM for simulation and TinyOS for deployment. Further the generated code is analyzed via network simulations and real-world deployment test. Through the process of MDSE-based code generation and analysis, the protocol design is validated, verified and analyzed. We use the GinMAC protocol as a running example to illustrate the design and development life cycle.

Keywords: Model-Based Development, Code-generation, Medium Access Control Protocols, Colored Petri Nets, Simulation, Implementation, Wireless Sensor Actuator Networks (WSAN)

1 Introduction

A wireless network of connected sensors and actuators operating to fulfill a specific collective goal constitutes a Wireless Sensor-Actuator Network (WSAN). The sensors and actuators are resource-constrained devices operating on batteries. Among several application domains, process automation and factory automation are important application areas in the industrial domain. Specifically, the application of a WSAN in control-loop automation is an important research area. These applications have strict real-time requirements and are in many cases safety critical (e.g., nuclear power plants). Thus, the design of solutions that include software for the network nodes is required to have sound design and development methodology to result in a verified and validated design that also satisfies the real time requirements. The classical design methodology is to: (1) outline the requirements to the solution; (2) design a solution based on the requirements; (3) carry an analytical evaluation of the performance of the solution; and (4) do a manual conversion of the design into simulation code for further performance analysis, and (5) convert the design into implementation code and perform deployment test on hardware.

The Dual-Mode Adaptive MAC protocol (DMAMAC) [10] is a Medium Access Control (MAC) protocol for process control applications. The traditional design methodology was employed for DMAMAC protocol design [10] based on application requirements and further evaluated analytically. The DMAMAC protocol was simulated and evaluated for performance [9]. Further, was evaluated with real-world deployment [11] (these steps correspond to 1,2,3,4 & 5). Three important issues that can arise in the general design methodology are: human induced errors in manual conversion, time consuming manual conversion, and the requirement to make manual changes at each step when changes are required to the design or the requirements. With the use of emerging software engineering practices, one can improve the design and development process. This could help in further strengthening the reliability of the software part of the solution, while additionally reducing the time from design to development, thus reducing the cost. Model-Driven Software Engineering (MDSE) [3] is one such approach that has long been seen as a prominent approach for software engineering. MDSE is currently used in several industrial application domains [6]. In this article, we attempt to create a MDSE approach with MAC protocols in focus. The DMAMAC protocol is rather complex compared to the GinMAC protocol upon which the DMAMAC protocol design is based. Thus, we use the GinMAC protocol as a basis to build the MDSE approach and then proceed towards applying the principle to the DMAMAC protocol as well.

In the MDSE approach, we start with an abstract platform independent representation of the solution, protocols for example. Abstraction allows for focusing on behaviour of the protocol. Using formal approaches on the abstract models allows for verification of the behaviour of the protocol via model checking or theorem proving. This abstract model can further be simulated to obtain an initial performance assessment. Thus, the protocol can be validated for performance requirements, and verified for software requirements. One tool that allows both model-checking and simulation is CPN Tools [7]. CPN Tools is based on the expressive Colored Petri Nets (CPN) language combined with the Standard ML programming language. Previously, CPN has been used for modeling and verification of network protocols [1]. Further, we use Petricode [20] tool for the semi-automatic code generation part. This forms the MDSE approach proposed previously in [12]. In this article, we extend this work to complete the code generation for the simulation platform and hardware platform. The generated code is used to analyze the protocol performance via network simulations on MiXiM, and via deployment in a real-world setting of the TinyOS code. We also discuss the methodology used to design the CPN model. We use GinMAC [21] protocol as a running example to present our MDSE approach.

Related Work. A model-based development approach has been applied in the WSAN domain [15, 22, 19]. Multiple works have proposed frameworks for rapid prototyping of the development model, mostly based on either, Domain Specific Modeling Languages (DSML) [2, 4] or the Unified Modelling Language (UML) [15, 22, 19]. In [16], the authors propose a design framework to convert models created in Simulink to platform specific code for the platforms TinyOS and MANTIS operating system. They also provide simulation and behavioural analysis. An Architectural framework for Wireless Sensor Actuator Networks (ArchWiSeN) was proposed in [18]. This is based on the generic modeling platform, UML, for abstract and platform independent representation of the models. Platform specific code generation is performed to obtain code for TinyOS, and simulated using the Micaz platform provided in the TOSSIM [14] simulator by TinyOS.

In this article, we choose to create the platform-independent abstraction of the model in CPN Tools. CPN tools allows state-space based verification, and simulation. We specifically focus on behavioural modeling of the protocol design. A typical WSN node implements an application protocol, a routing protocol, a MAC protocol, and a link layer protocol. Thus, the solution in its entirety is made up of multiple protocols, making up a complex solution. In [16, 18], the authors view the solution as multiple nodes depicting common WSN behaviour. Further, in this article, we provide platform specific code generation for MiXiM, a network simulator specifically built for wireless networks. Also, the deployment specific code generation targets the TinyOS platform and is tested via deployment on Zolertia Z1 [23] motes.

The rest of the paper is divided into five sections. We revisit our model-based development approach and PetriCode [20] in Section 2. Also, the GinMAC protocol [21] which is used as an example is introduced. In Section 3, we describe the CPN model of the GinMAC protocol. The code generation templates, pragmatics used, and the generated code for MiXiM-OMNeT are discussed in Section 4. Code generation for TinyOS is discussed in Section 5. Finally, in Section 6, we sum up conclusions and discuss future work. The article assumes prior knowledge of Petri nets. The MDSE approach and Model-based Development approach are used interchangeably, and means the same in this context.

2 MDSE and PetriCode

The MDSE approach is shown in Fig. 1. Compared to the MDSE approach presented in the previous article [12], we have realized the work in progress (now the working part), and have implemented and analyzed the generated code via simulation and deployment. This includes the design of an example

MAC protocol, code generation for the simulation platform MiXiM [8] based on OMNeT, and for the operating system TinyOS [13] for hardware platforms. The GinMAC [21] protocol is used as a running example for the code generation case study and is introduced later in this section. Following the MDSE approach the GinMAC protocol is first designed using CPN Tools. Initial design verification, validation, state-space analysis, and simulation are performed using CPN Tools. Further, platform dependent code generation is performed to obtain code for MiXiM and TinyOS. The code generation is done using the PetriCode tool as described below.

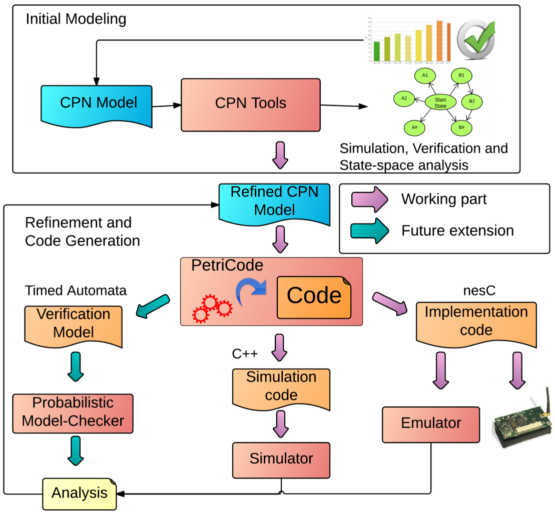


Fig. 1. MDSE approach for protocol design [12]

PetriCode. We use the PetriCode [20] tool for the code generation. PetriCode is a template based code generation tool designed to transform a subclass of CPN models called Pragmatic Annotated-CPNs (PA-CPN) to implementations. PA-CPN models enforce a hierarchical structure on the models with three levels. The top level module in a PA-CPN model is called the Protocol System Module (PSM). The PSM contains the principal agents of the protocol and the channels between them. Each principal in the PSM contains sub-modules that are on the principal level. The principal level modules (PLMs) contain the services that are provided by the principal as well as places that are common among the services of the principal, and life-cycle variables that control when services can be invoked. Each service in the PLMs has sub-modules on the service level. Service level modules (SLMs) contain the actual behaviour of each service. SLMs are further decomposable into control-flow blocks that represent structures such as loops and conditionals. In addition to enforcing a hierarchical structure, PA-

CPNs allow model elements to be annotated with code generation pragmatics. The pragmatics allow the code generator to identify the elements of the PA-CPN model and to find appropriate templates for the code generation. The code generation process in PetriCode starts by reading and parsing a PA-CPN model. Then the PA-CPN model is transformed into an intermediary representation called an Abstract Template Tree (ATT). The ATT mirrors the hierarchical structure of a PA-CPN model. PetriCode exploits the pragmatics, to decide what templates to execute at each node in the ATT. The generator executes templates for each of the nodes in the ATT. Finally, the code generated for the ATT nodes are combined to obtain the complete code for each principal. In this article, we design and develop separate templates for the MiXiM and TinyOS platforms. Based on these templates the final platform dependent code is obtained.

Code Generation Goals. The code generation phase assists in reducing errors induced by humans in programming and streamlines the implementation approach based on a modular implementation approach. Also, the conversion to simulation platform assists in analyzing the design of the protocol in the simulated world to assess the performance. Network simulators specialized in wireless simulations offer emulated environments of wireless channels, energy consumption, data transmission, and other services. The conversion to MiXiM code for simulation has two advantages: firstly, performance analysis of the designed protocol; secondly, comparison of the protocol with existing protocols based on selected performance metrics. The conversion to TinyOS supporting Network Embedded Software C (nesC) code allows the protocol to run across multiple hardware components that support the TinyOS platform. This also allows the users to validate the operation of their protocol on existing hardware, and provides opportunities for further testing in a real environment.

2.1 The GinMAC Protocol

GinMAC is a Time Division Multiple Access (TDMA) protocol, proposed in the GINSENG project [17]. GinMAC was developed to address the real-time requirements for industrial monitoring and control applications. A typical GinMAC superframe is shown in Fig. 2(b). A Finite State-Machine (FSM) representation of a node working on the GinMAC protocol is shown in Fig. 2(a). The main features of the protocol are: *Offline Dimensioning*, *Exclusive TDMA*, and *Delay Conform Reliability Control*. *Offline Dimensioning* means that deployment and delay requirements are planned before deployment, and all scheduling decisions are made offline prior to deployment. *Exclusive TDMA* means that all TDMA slots are exclusive and are not re-used. *Delay Conform Reliability Control* means that GinMAC uses reliability control mechanisms in the form of additional re-transmission slots to increase reliability. The number of additional slots required are calculated based on the wireless channel conditions in the area of deployment. The GinMAC protocol is designed for networks having a tree topology. It has three types of slots: *basic*, *additional*, and *unused*. *Basic* slots are for regular sensor and actuator data. The *Additional* slots feature is intended to increase robustness in poor channel conditions. *Unused* slots are sleep slots for duty cycling and energy conservation.

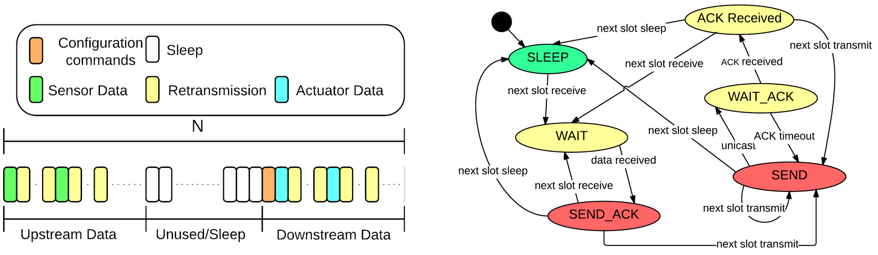


Fig. 2. The GinMAC superframe (a) and the GinMAC Finite State Machine (b)

3 CPN Model

As a first step we create a CPN model for the GinMAC protocol. The top level module (Protocol System Module) of a sensor/actuator node within the GinMAC protocol is shown in Fig. 3. The platform independent CPN model includes an application/network module, the GinMAC module, a radio module, and a wireless channel module for data exchange. The pragmatics assisting in the code generation are by convention written inside <<>>.

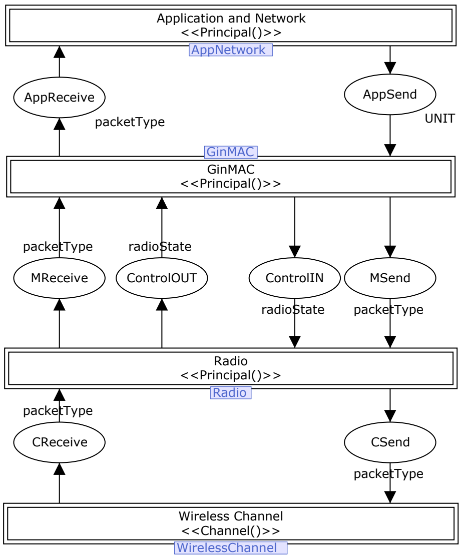


Fig. 3. Top-level CPN module

3.1 The GinMAC Module

We use a modular modeling approach for design. Different MAC protocols share common features that can be designed as basic re-usable components which further increases the re-usability of protocol design and also the code generation approach. The abstract model in CPN is platform independent, hence the model can be used to generate code-specific to different platforms as well. The re-usability

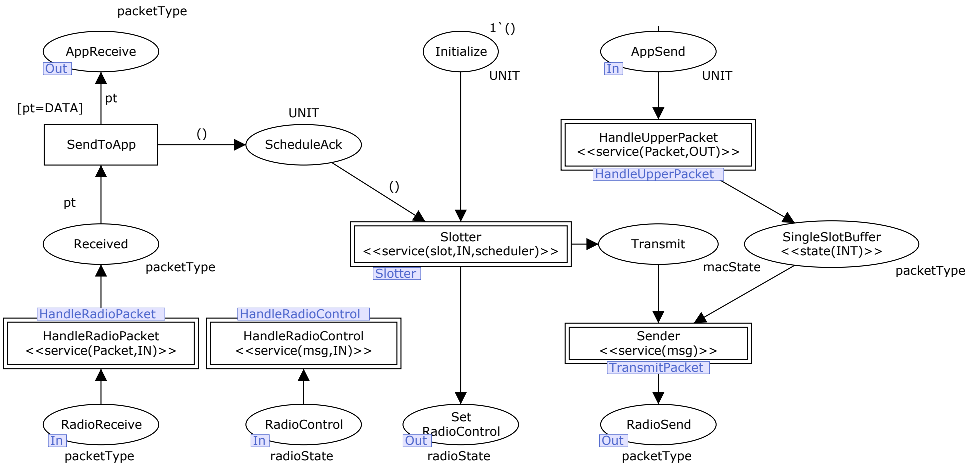


Fig. 4. CPN model of a MAC protocol

allows different MAC protocols to be modeled and code to be generated for the same platform. The detailed GinMAC layer in the principal (second) level is shown in Fig. 4. This model contains the GinMAC functions including: send/receive data packets, slot scheduler, and control packets handler. The send/receive of data packets facilitates the data transfer for the application through the entire network and is handled by two services: Sender and HandleRadioPacket. The radio control packet handling is local to the node and is handled by the service HandleRadioControl. The slot scheduler manages the operation to be performed at the node at each instant of time based on the superframe. The slot scheduler service is handled by Slotter. The main set of operations in GinMAC also called as MAC states includes: send, wait (receive), and sleep as depicted in the FSM in Fig 2(a). These operations of the GinMAC protocol are grouped as services which are defined in more detail in the service level of the PetriCode approach. The operations defined here are also basic MAC operations for other protocols including the DMAMAC protocol, and hence can be re-used.

3.2 The Slotter Function

Fig. 5 shows the *slotter* function of the protocol used to implement different slots based on the timer. These slots represent one of the operations to be performed. Since GinMAC is a TDMA-based protocol, a timer is used to run through the slots. The *slotter* function calls services based on the operations: send (SEND_DATA/ACK), receive (WAIT_DATA/ACK), and sleep, appropriately depending on the time. Based on the slot type, an appropriate service is called to handle the operation: Receive, Transmit, or Sleep. In Fig. 5, we have omitted send and wait for notifications to keep the figure simple and small, but the design of notification slot is included in the complete model and in the code generation templates. Notifications are an alternative representation of the configuration commands used by GinMAC protocol, a generic concept used by MAC protocols to send network wide updates.

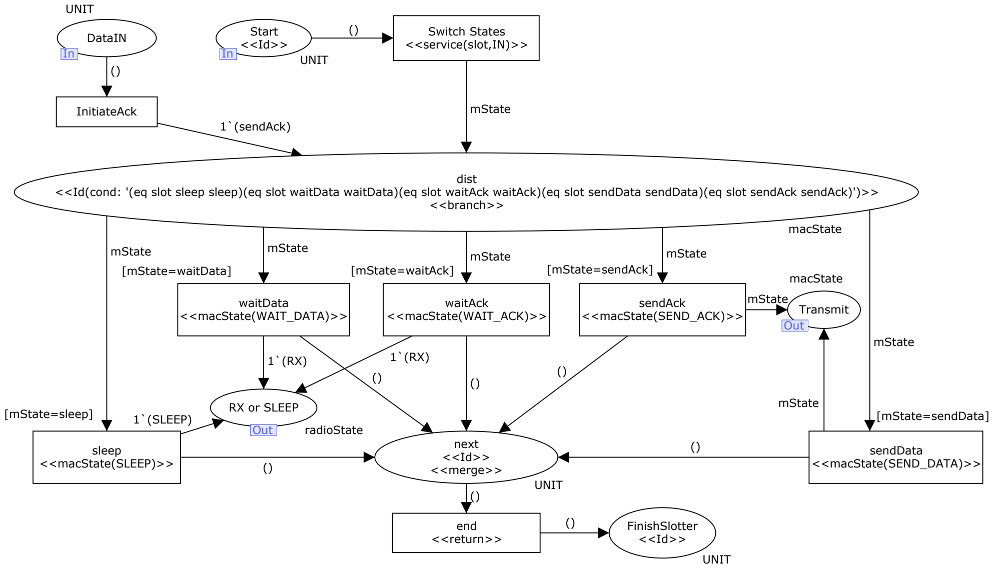


Fig. 5. Slotter function of the MAC protocol

One of the services used by Slotter, the Sender service for transmitting packets is shown in Figure. 6. The service handles all outgoing packet types including data, ack, and notification packets. It sends the packet to the radio service for further handling of the physical transmission.

4 MiXiM Code Generation

OMNeT++ is graphical modeling framework along with discrete event-based simulation and analysis extensions with graphical user interface support. MiXiM [8] is a modeling framework, which is a result of integrating several OMNeT++ frameworks, designed specifically for simulating mobile and fixed wireless networks. MiXiM is a modular framework consisting of pre-installed implementation code for radio modules, and base modules for application, network (routing), and the MAC layer. The workflow for simulation-analysis including the node software architecture in MiXiM is shown in Fig. 7.

MiXiM simulations are specified using a combination of source files, NED files (Network Descriptors), configuration.ini (configuration file), and other XML files describing the physical attributes of the network. The source C++ files are mainly used describe the protocols at each layer. The NED files are used to

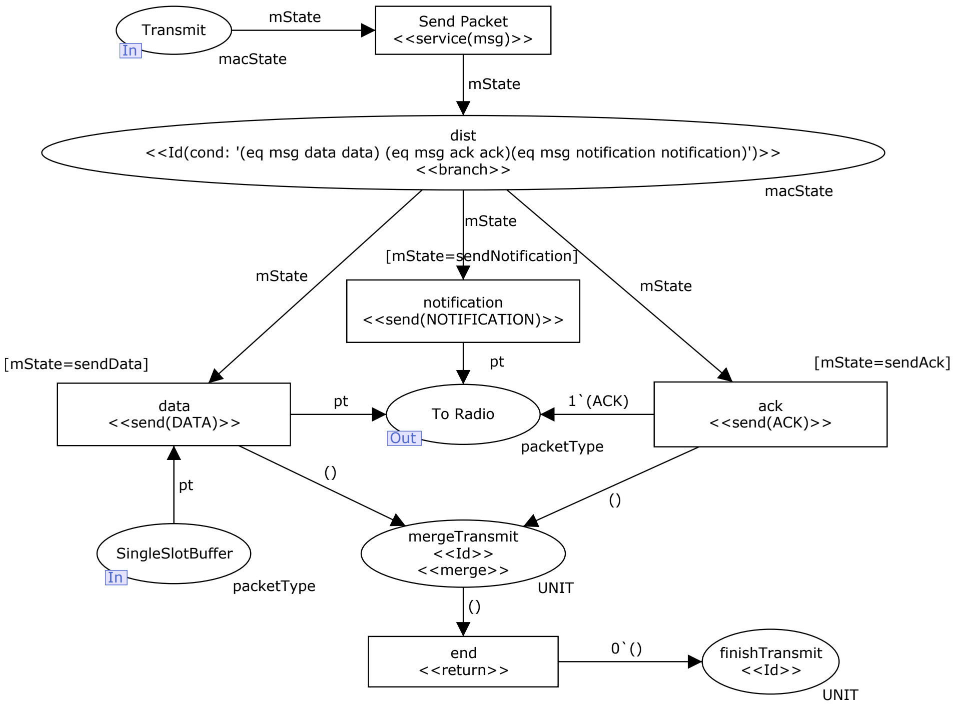


Fig. 6. Transmit Packet function of the GinMAC protocol

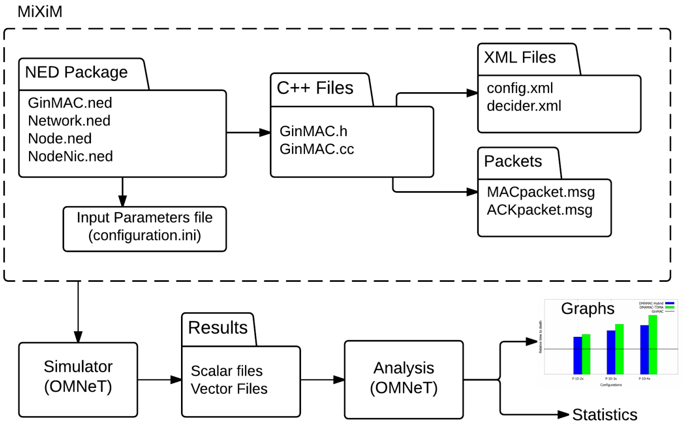


Fig. 7. MiXiM node software architecture with simulation and analysis flow

describe each type of physical node in the network, the network configuration defining how these nodes connect with each other, and the user defined module to be used in these nodes. The configuration.ini file allows the user to define explicitly various parameters concerning each layer (physical, MAC, application and network). It also includes position data for nodes (if required) and multiple configurations for simulation. Further, XML files are used to define the path-loss models, packet loss, signal to noise ratio (SNR), and decider configurations. These features are specific to the MiXiM simulation platform and are generated for basic configuration using the code generation templates, and are not a part of the CPN model. Apart from the generation of the MAC source files, NED files for basic network layer configuration, basic configuration files, and network configuration XML files are also generated.

4.1 MiXiM MAC model

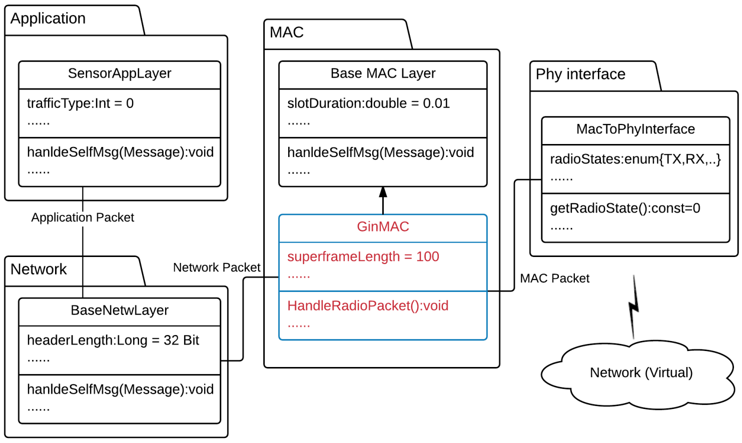


Fig. 8. An abstract UML view of the source code for a node

The focus is to generate the MAC layer implementation code, and hence we use the existing modules of the MiXiM framework to construct the complete implementation. Creating a new module in MiXiM includes implementing the base functions for that module (BaseMacLayer) and extending it with the protocol specific functions. We use the code templates to generate the protocol with methods extended from the BaseMacLayer. The generated MiXiM source code from the protocol model is then used as the MAC protocol, and other modules (application/routing) are used to complete the node software architecture to make it simulation ready. An abstract UML representation of the generated source code (GinMAC) put in context along with the existing modules is shown in Fig 8. The GinMAC source code generated for MiXiM inherits from the BaseMacLayer, and extends it with its functions and characteristics. The functional view of the GinMAC protocol is shown in Fig. 9. The functional view corresponds to the principal level of the CPN model shown in Fig. 4. The MAC protocol handles application/network packets and incoming packets from other nodes, and sends them over the radio channel to a corresponding node based on a pre-decided schedule determined by the GinMAC superframe structure shown in Fig. 2(b).

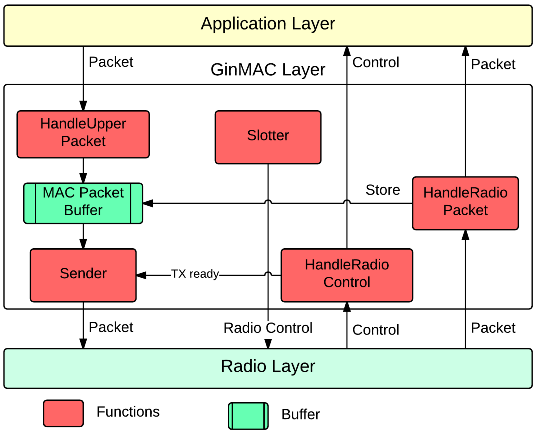


Fig. 9. A graphical representation of the functions in the MiXiM MAC protocol

4.2 PetriCode

The MAC protocol designed in CPN is used to generate a MiXiM equivalent. We use the PetriCode tool [20] for the code generation. The PetriCode tool, based on the code generation templates developed for the MiXiM platform, generates the required C++ source code. An example template for the GinMAC states SEND_DATA, WAIT_DATA, and SLEEP are presented in List. 1.1. The main GinMAC states are: SEND_DATA, SEND_ACK, SEND_NOTIFICATION, WAIT_DATA, WAIT_ACK, WAIT_NOTIFICATION and SLEEP. A MAC state represents the type of operation being performed at a given time. Each of the MAC states have an associated code template for generation. Based on these code templates, we obtain the main source code. The code template example for MAC states is shown in List. 1.1. The params variable in the template checks for the parameters within the keyword (macState) for the associated template, then generates the code for it. The generated C++ source code for the Slotter function shown in Fig. 5 is presented in Listing 1.2. Three of the MAC states SEND_DATA, WAIT_DATA and SLEEP have been presented along with the generated code. In the full version implemented, all the MAC states have their respective code templates as opposed to only three shown here.

Listing 1.1. PetriCode C++ code generation template

```
Pragmatic : <macState(SLEEP)>
Template code for MiXiM:
<%if(params[0].toString() == "SLEEP")
{%> currentMacState = SLEEP;
    debugEV << "Going to Sleep" <<endl;
    phy->setRadioState(MiximRadio::SLEEP);
    /* @brief Finds the next slot after getting up */
    findDsitantNextSlot();
<%}

Pragmatic : <macState(WAIT_DATA)>
Template code for MiXiM:
<%if(params[0].toString() == "WAIT_DATA")
{%> currentMacState = WAIT_DATA;
        debugEV << "My receive slot" <<endl;
        phy->setRadioState(MiximRadio::RX);
        debugEV << "Switching Radio to receive mode" << endl;
        /* @brief Procedure for finding nextSlot */
        findImmediateNextSlot(currentSlot, slotDuration);
        currentSlot++;
        currentSlot %= numSlots;
<%}

Pragmatic : <macState(SEND_DATA)>
Template code for MiXiM:
<%if(params[0].toString() == "SEND_DATA")
{%> currentMacState = SEND_DATA;
    if (mySlot == transmitSlot[currentSlot]){
        if(macPktQueue.empty()){
            debugEV << "No Packet to Send exiting" << endl;
            findNextSlot(currentSlot, slotDuration);
    }else{
        phy->setRadioState(MiximRadio::TX);
        debugEV << "Waking up in my slot.
            Radio switch to TX command Sent" << endl;
        findImmediateNextSlot(currentSlot, slotDuration);
    }
    currentSlot++;
    currentSlot %= numSlots;
<%}
```

Listing 1.2. C++ source code

```
//GinMAC C++ file
void GinMAC::Slotter(message_t msg){
if( msg == sleep){
    currentMacState = SLEEP;
    debugEV << "Going to Sleep" <<endl;
    phy->setRadioState(MiximRadio::SLEEP);
    /* @brief Finds the next slot after getting up */
    findDsitantNextSlot();
}
else if( msg == waitData){
    currentMacState = WAIT_DATA;
    debugEV << "My receive slot" <<endl;
    phy->setRadioState(MiximRadio::RX);
    debugEV << "Switching Radio to receive mode" << endl;
    /* @brief Procedure for finding nextSlot */
    findNextSlot(currentSlot, slotDuration);
    currentSlot++;
    currentSlot %= numSlots;
}
else if( msg == sendData){
    currentMacState = SEND_DATA;
    if (mySlot == transmitSlot[currentSlot]){
        if(macPktQueue.empty()){
            debugEV << "No Packet to Send exiting" << endl;
            findNextSlot(currentSlot, slotDuration);
    }else{
        phy->setRadioState(MiximRadio::TX);
        debugEV << "Waking up in my slot.
            Radio switch to TX command Sent" << endl;
        findImmediateNextSlot(currentSlot, slotDuration);
    }
    currentSlot++;
    currentSlot %= numSlots;
}}
```

4.3 MiXiM Simulation

The MiXiM simulation engine allows for simulation, and statistics collection, which can be further used to generate graphs. The graphs support the assessment of the performance of the protocol. Statistics can be collected in either scalar or vector form. For our simulation test we used the scalar forms. We performed test simulation on a small network with 7 nodes and 1 sink. The node configuration used for the study is shown in Fig. 10. We used the CC2420 radio decider module, the CC2420 radio power consumption values, and switch times to obtain accurate results.

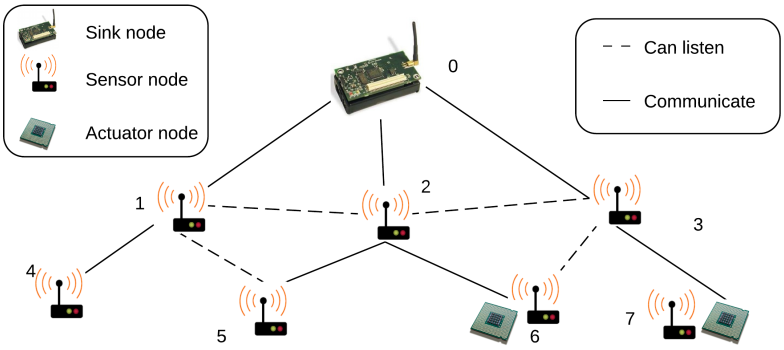


Fig. 10. The network topology used for simulation

The resulting graph from counting the number of packets transmitted and received for each node is given in Fig. 11. Also, a graph with network lifetime based on a given initial battery capacity is shown in Fig. 11. The lifetime is

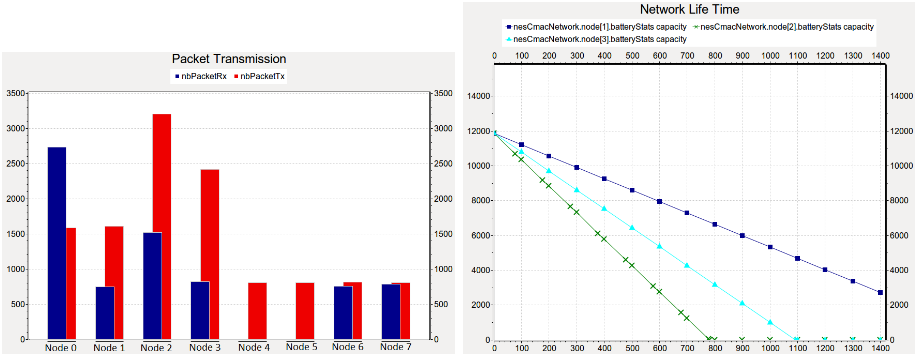


Fig. 11. The packet reception and transmission statistics for the GinMAC protocol (a) and Network life time in terms node deaths (b)

related to the starting capacity. Given the load, it is evident that node 2 depletes its energy first among all the nodes in the network and thus is the first to die as shown in Fig. 11. The data transmission graph shows the amount of data transmission being handled for a given simulation duration, 1500 seconds for our case. In the process two nodes die, node 2 and node 3.

5 TinyOS nesC Code Generation

TinyOS is one of the commonly used operating systems for resource limited WSAN hardware implementations. nesC is a component-based programming language for the TinyOS platform. It mainly consists of *components* which are represented using *modules* and *configurations*. These components generally provide services to other components and use functions provided by other components via a set of *interfaces*. The implementation part uses *commands*

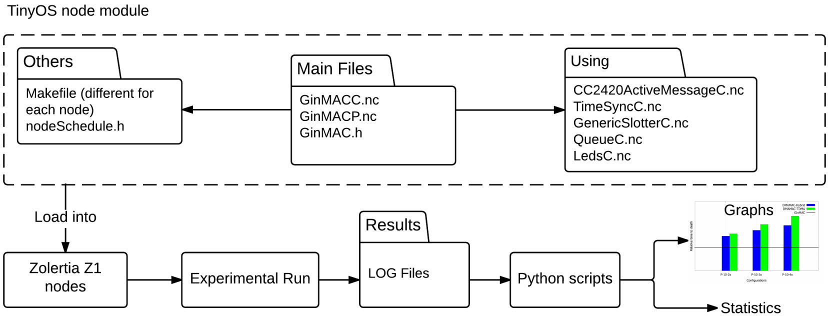


Fig. 12. nesC implementation, deployment and analysis flow

and *events* which are called or signaled, respectively. *Commands* are used to define operations that can be triggered. *Events* represent hardware events that is similar to an interrupt to indicate an event occurring, e.g. reception of a packet. *Tasks* are also a part of the programming language which can be called from both events and commands. The workflow for implementation on TinyOS using the generated nesC source code, and further performing deployment and analysis is shown in Fig. 12. For evaluation of the generated nesC code the presented workflow is used. The generated source code along with an application is compiled for the platform hardware Zolertia Z1 [23]. Furthermore, based on pre-decided topology, schedule, and location, we conducted an experiment. The collected results are placed in a log file, which is used by python scripts to generate graphs and statistics. The generated GinMAC source code uses multiple off the shelf components to complete the solution, e.g., the CC2420ActiveMessageC radio module provided by TinyOS.

5.1 MAC nesC Model

The component graph of the GinMAC nesC module is shown in Fig. 13. The GinMAC protocol uses the radio functions provided by the existing component in TinyOS, the CC2420ActiveMessageC component. For time synchronization, an essential function required for Time Division Multiple Access (TDMA) protocols like GinMAC, we use the TimeSyncC component. The TimeSyncC provides the Flooding Time Synchronization Protocol (FTSP). As a scheduler, the GinMAC component uses the GenericSlotterC component. The scheduler goes through the superframe structure and executes the corresponding slot for the given instance. The component QueueC is used to create a packet buffer for incoming packets to be forwarded. The incoming arrows to the GinMAC module are the features that are provided by the GinMAC module to the application or the network modules that use GinMAC. In the current version, we generate code that uses the radio layer in an abstract form. Whereas a more detailed and explicit control of radio can be implemented similar to the MiXiM code. This is detailed in the implementation of MAC layers in TinyOS 2 [5]. With such explicit control, a MAC protocol can control the transmission power at which each packet is sent.

5.2 PetriCode

Similar to the MiXiM code generation, the GinMAC protocol designed in CPN is used for nesC code generation targeting TinyOS. PetriCode, based on the templates defined for nesC code generation, generates the nesC code. We present the same three GinMAC states and the template for code generation as in MiXiM. The GinMAC states SLEEP, WAIT_DATA and SEND_DATA are presented along with the employed code generation templates as shown in List. 1.3. The generated nesC source code for the Slotter function shown in Fig. 5 is presented in List. 1.4.

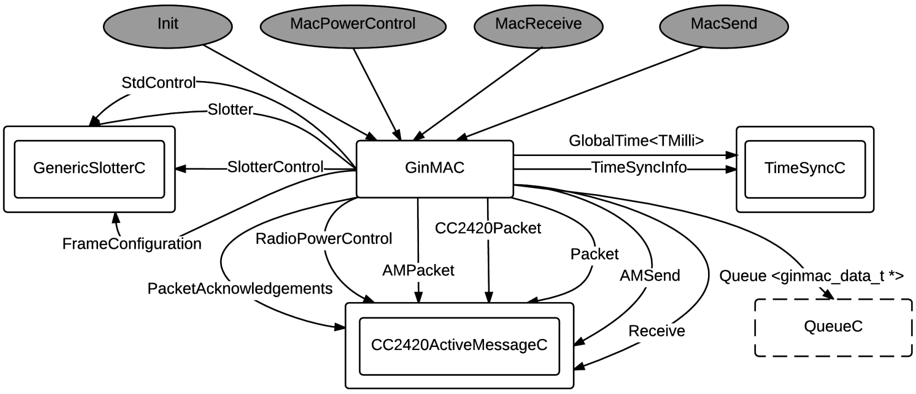


Fig. 13. GinMAC protocol nesC component model

Listing 1.3. PetriCode nesC Template

```
Pragmatic: macState<SLEEP>
Template code for nesC:
if(params[0].toString() == "SLEEP")
{%>     currentMacState == "SLEEP";
        if(!radioOff){
                printfz1("Calling radio sleep");
                call RadioPowerControl.stop();
                call Leds.led0Off();
}<%}%>

Pragmatic: macState<WAIT_DATA>
Template code for nesC:
if(params[0].toString() == "WAIT_DATA")
{%>     currentMacState == "WAIT_DATA";
        printfz1("Waiting for Data");
        /* @brief Switching on Radio if OFF, there should be an event if packet is arrived */
        if(radioOff){
                call RadioPowerControl.start();
                call Leds.led0On();
}<%}

Pragmatic: macState<SEND_DATA>
Template code for nesC:
else if(params[0].toString() == "SEND_DATA")
{%>     currentMacState == "SEND_DATA";
        /* @brief Checking if the slot is the node's transmit slot  */
        data = (data_t*)call Packet.getPayload(&dataPkt,sizeof(data_t));
        data->nodeId = TOS_NODE_ID;
        data->destinationId = sinkId;
        data->dataSeqNo = dataSeqCount;
        if (phyLock == FALSE){
                if(radioOff){
                        /* @brief Switching on the Radio */
                        printfz1("Waking up");
                        call RadioPowerControl.start();
                        call Leds.led0On();
                }
                call ACK.requestAck(&dataPkt);
                /* @brief Sending Data via radio/phy interface */
                if(call PhySend.send(parentId[TOS_NODE_ID],
            &dataPkt,sizeof(data_t)) == SUCCESS){
                        atomic phyLock = true;
                        post taskPrint(data);
}}<%}
```

Listing 1.4. nesC source code

```
//necC code generated for slotter service
event void Slotter.fire(uint8_t slot)
{
if(slot == sleep){
    currentMacState == "SLEEP";
        if(!radioOff)
        {       printfz1("Calling radio sleep");
                call RadioPowerControl.stop();
                call Leds.led0Off();
}}
else if(slot == waitData){
    currentMacState == "WAIT_DATA";
    printfz1("Waiting for Data");
    if(radioOff){
        call RadioPowerControl.start();
        call Leds.led0On();
}}
else if(slot == sendData){
    currentMacState == "SEND_DATA";
        /* @brief Checking if the slot is the node's transmit slot  */
        data = (data_t*)call Packet.getPayload(&dataPkt,sizeof(data_t));
        data->nodeId = TOS_NODE_ID;
        data->destinationId = sinkId;
        data->dataSeqNo = dataSeqCount;
        if (phyLock == FALSE){
                if(radioOff){
                        /* @brief Switching on the Radio */
                        printfz1("Waking up");
                        call RadioPowerControl.start();
                        call Leds.led0On();
                }
                call ACK.requestAck(&dataPkt);
                /* @brief Sending Data via radio/phy interface */
                if(call PhySend.send(parentId[TOS_NODE_ID],
            &dataPkt,sizeof(data_t)) == SUCCESS){
                        atomic phyLock = true;
                        post taskPrint(data);
}}}
```

5.3 Implementation Evaluation

We evaluated the generated nesC source code on a hardware platform (Zolertia Z1 [23]) as mentioned in the nesC work flow shown in Fig. 12. Zolertia nodes are powered by an MSP430F2617 low power microcontroller with 16-bit RISC CPU operating at 16 MHz clock speed. It also packs in 92KB flash memory and 8KB RAM. The node is IEEE 802.15.4 compliant and uses a CC2420 transceiver operating at 2.4GHz with a data rate of 250 Kbps. Contrary to the simulation experiments, we performed certain link quality measurements using the hardware platform. We use a smaller topology than the one

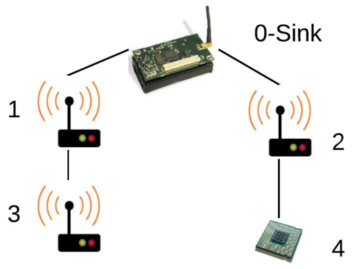


Fig. 14. The network topology used for the deployment analysis

used in MiXiM shown in Fig. 14 to keep the
experiment simple. We deployed these nodes in a corridor with each node placed at a distance of 5m from its parent. Thus, farthest nodes are 10m away from the sink. The corridor also had constant movement of humans. For implementation based evaluation we focused on two new metrics and mainly performed link quality assessment for presentation of basic results. Two metrics used for the assessment are Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI). The RSSI measured for the nodes in the network is shown in Fig 15 (left) and the link quality is shown in Fig. 15 (right). RSSI indicates the strength at which the receiver receives the signal (range -45dBm to -95 dbm (lowest possible)). The LQI is a calculated assessment of the link quality based on lost bits in received packets given by TinyOS, the value ranges from 50 to 110, with values towards 110 considered to be optimal. These values were obtained from the built in TinyOS functions provided by the interface CC2420Packet. The obtained RSSI graph shows varying levels of RSSI for the nodes, which is generally based on the distance between the sender and the receiver. RSSI is also affected by interference, mainly from people moving through the corridor time to time and also, the WiFi service operating in the vicinity. LQI levels mostly lie in between 95-110 indicating the link quality is good between the nodes despite of the RSSI differences. The LQI also falls very low for one of the nodes between rounds 50 and 100 (node 4), this might be caused by continuous packet failure during a period. Further reasons for this are not provided in the basic analysis provided here since it is beyond the scope of this article. However, this is an important result since it shows the possible variation in a real environment. The LQI and RSSI values combined give a collective picture of link quality between two nodes. If both RSSI and LQI values are considered for node 4 that is affected between rounds 50 and 100 in LQI, the RSSI graph also has correspondingly low readings for those same rounds.

6 Conclusions and Future Work

Model-driven software engineering is a popular approach for design and development of general computing applications. We have used MDSE principles and applied it to protocol design and development in the WSN domain. In this article, we have used model-based development techniques to generate code for two different platforms: simulation and deployment, from a CPN model of the GinMAC protocol. We have used the PetriCode tool for code generation. We developed templates for MiXiM, a wireless network simulator platform, and TinyOS, an operating system for hardware platforms. We have also analyzed the generated program code to present some performance evaluations that can be obtained based on the generated code. We performed separate analyses on these two: energy consumption and lifetime on the MiXiM simulator platform, and link quality assessment using Zolertia nodes operating on TinyOS code. One important comparison between the classical methodology used for the DMAMAC protocol to the MDSE methodology used here is that the generated platform specific models are closely linked to the CPN model, thus provide a

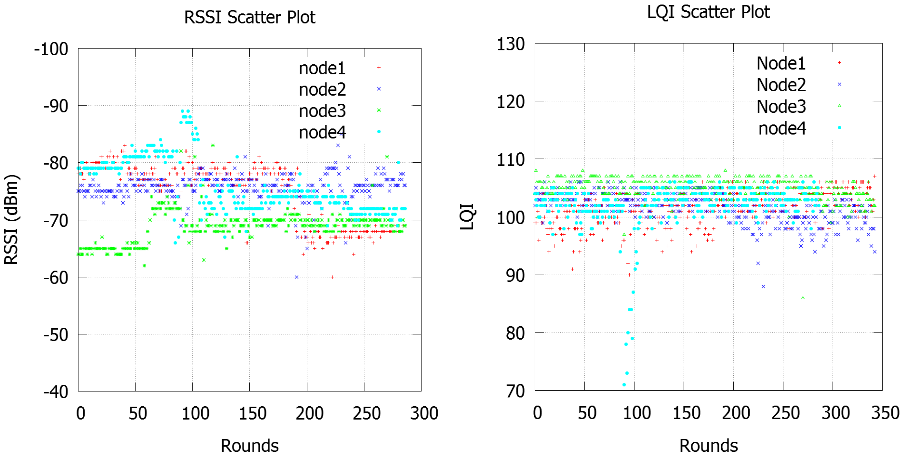


Fig. 15. Link quality based on RSSI (a) and LQI (b)

higher confidence in the generated code. The models in each step of the classical methodology on the other hand, are entirely based on requirement specification and no direct conversions are made.

In terms of future work and extension, we would like to mainly extend the code generation to specialized formal analysis tools like Uppaal or PRISM, which allows for further validation, and verification of real-time requirements of the protocols. Also, to extend the code generation to multiple platforms by exploiting their similarity with the existing code generation templates. Importantly, this applies to Castalia for simulation and Contiki OS for deployment. Castalia, is another wireless network simulation framework based on OMNeT++ and shares similarities with MiXiM. Contiki OS, an event-based operating system similar to TinyOS is an emerging operating system for low power sensor hardware, and is gaining market share rapidly. Apart from this, we would like to apply the MDSE approach to the DMAMAC protocol requirements to validate its usability.

References

1. Billington, J., Gallasch, G., Han, B.: A Coloured Petri Net Approach to Protocol Verification. In: Lect. on Concurrency and Petri Nets, pp. 210–290. Springer (2004)
2. Boonma, P., Suzuki, J.: Model-driven performance engineering for Wireless Sensor Networks with feature modeling and event calculus. In: Proceedings of the 3rd BADS Workshop. pp. 17–24 (2011)
3. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice. Synthesis Lectures on Software Engineering 1(1), 1–182 (2012)
4. Doddapaneni, K., Ever, E., Gemikonakli, O., Malavolta, I., Mostarda, L., Muccini, H.: A model-driven engineering framework for architecting and analysing Wireless Sensor Networks. In: Proceedings of the 3rd SESENA Workshop. pp. 1–7 (2012)
5. Hauer, J.H.: Tkn15. 4: An ieee 802.15. 4 mac implementation for tinyos. TKN Technical Reports Series (2009)

6. Hutchinson, J., Rouncefield, M., Whittle, J.: Model-driven engineering practices in industry. In: Proceedings of the ICSE. pp. 633–642 (May 2011)
7. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN tools for modelling and validation of concurrent systems. International Journal of Software Tools for Technology Transfer 9, 213–254 (2007)
8. Köpke, A., Swigulski, M., Wessel, K., Willkomm, D., Haneveld, P.T.K., Parker, T.E.V., Visser, O.W., Lichte, H.S., Valentin, S.: Simulating wireless and mobile networks in OMNeT++ the MiXiM vision. In: Proceedings of SIMUTools (2008)
9. Kumar S., A.A., Kristensen, L.M., Øvsthus, K.: Simulation-based Evaluation of DMAMAC: A Dual-mode adaptive MAC Protocol for Process Control. In: Proceedings of the 8th SIMUTools Conference. pp. 218–227 (2015)
10. Kumar S., A.A., Øvsthus, K., Kristensen, L.M.: Towards a dual-mode adaptive MAC protocol (DMA-MAC) for feedback-based networked control systems. Procedia Computer Science 34, 505–510 (2014)
11. Kumar S., A.A., Øvsthus, K., Kristensen, L.M.: Implementation and Deployment Evaluation of the DMAMAC Protocol for Wireless Sensor Actuator Networks. In: Proceedings of the 7th ANT Conference. vol. 83, pp. 329–336 (2016)
12. Kumar S., A.A., Simonsen, K.I.F.: Towards a model-based development approach for Wireless Sensor-Actuator Network Protocols. In: Proceedings of CyPhy. pp. 35–39 (2014)
13. Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., Culler, D.: TinyOS: An operating system for sensor networks. In: Ambient Intelligence, pp. 115–148. Springer (2005)
14. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In: Proceedings of the 1st International Conference on Embedded networked sensor systems. pp. 126–137. ACM (2003)
15. Losilla, F., V Chicote, C., Alvarez, B., Iborra, A., Sanchez, P.: Wireless Sensor Network application development: An architecture-centric MDE approach. In: Software Architecture, LNCS, vol. 4758, pp. 179–194. Springer (2007)
16. Mozumdar, M.M.R., Gregoretti, F., Lavagno, L., Vanzago, L., Olivieri, S.: A framework for modeling, simulation and automatic code generation of Sensor Network Application. In: Proceedings of SECON. pp. 515–522 (2008)
17. O Donovan, T., Brown, J., Roedig, U., Sreenan, C., do O, J., Dunkels, A., Klein, A., Silva, J., Vassiliou, V., Wolf, L.: GINSENG: Performance control in Wireless Sensor Networks. In: Proceedings of the 7th SECON Conference. pp. 1–3 (2010)
18. Rodrigues, T., Batista, T., Delicato, F., Pires, P., Zomaya, A.: Model-driven approach for building efficient Wireless Sensor and Actuator Network applications. In: Proceedings of the 4th SESENA Workshop. pp. 43–48 (2013)
19. Shimizu, R., Tei, K., Fukazawa, Y., Honiden, S.: Model driven development for rapid prototyping and optimization of Wireless Sensor Network applications. In: Proceedings of the 2nd SESENA Workshop. pp. 31–36. ACM (2011)
20. Simonsen, K.I.F., Kristensen, L., Kindler, E.: Generating protocol software from cpn models annotated with pragmatics. In: Formal Methods: Foundations and Applications, LNCS, vol. 8195, pp. 227–242. Springer (2013)
21. Suriyachai, P., Brown, J., Roedig, U.: Time-critical data delivery in Wireless Sensor Networks. In: Proceedings of DCOSS. vol. 6131, pp. 216–229 (2010)
22. Vicente-Chicote, C., Losilla, F., Alvarez, B., Iborra, A., Sanchez, P.: Applying MDE to the development of flexible and reusable Wireless Sensor Networks. International Journal of Cooperative Information Systems 16(3-4), 393–412 (2007)
23. Zolertia: Z1 datasheet. `http://www.zolertia.io//`, accessed: 03-10-2015

List of Figures

List of Tables