Applied Formal Methods in Wireless Sensor Networks

zur Erlangung des akademischen Grades eines

DOKTORS DER NATURWISSENSCHAFTEN

der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)
genehmigte

Dissertation

von

Frank Werner

aus Lebach

Tag der mündlichen Prüfung: 22.07.2009
Erster Gutachter: Prof. Dr. P. H. Schmitt, Universität Karlsruhe (TH)
Zweiter Gutachter: Prof. Dr. F. Freiling, Universität Mannheim

"It is fair to state, that in this digital era correct systems for information processing are more valuable than gold."

H. BARENDREGT: The Quest for Correctness
Images of SMC Research 1996, pages 39-58

Acknowledgements

This thesis is a result of three years work of research during which I had the chance to meet new and formerly unknown people that influenced many thoughts and gave a concrete shape to many bright ideas.

Most of all I am deeply grateful to my advisor Prof. Dr. Peter H. Schmitt, who opened me the door to research with many challenging missions, various paths to be traversed, many things to learn, and constantly broadening my horizon. His vision of formal methods and their application to real-world scenarios gave me constant motivation and never exhausting ambition.

I want to thank my second reviewer Prof. Dr. Felix Freiling from the University of Mannheim for his positive response and feed back to my work.

Much of this success is owned to my colleagues from the chair of Logic and Formal Methods as they were good listeners for questions and problems. I especially appreciate the kind help of Dr. Steffen Schlager and Dr. Richard Bubel during the first year of my work. But I also want to thank Christian Engel, David Faragó, Mattias Ulbrich, and Benjamin Weiß for the pleasant work atmosphere and appealing discussions.

I thank the members of the ZeuS project for their kind cooperation and the support that I received when working on practical aspects of sensor networks. Foremost to name Simon Kellner, Joachim Wilke, Dr. Zina Benenson, Dr. Erik-Oliver Blaß, and all the others that truly gave inspiration and prospect to this work.

In the end I like to express my gratitude to my parents who supported my education over all those years. Without them all of this would have never worked out. Finally I want to thank Andrea with all my heart for her overall moral support and patience. Thank you to all of you!

Karlsruhe, July 2009
Frank Werner

Zusammenfassung

Unter der Anwendung Formaler Methoden versteht man den Einsatz mathematischer Modelle und Techniken zur Analyse von Informations- und Kommunikationstechnologien. Modellprüfverfahren sind ein Teilgebiet hiervon und erlauben dem Benutzer eine vollständige Untersuchung des Suchraums der vom modellierten System aufgespannt wird mittels *brute-force* Suche. Analysen beschränken sich hierbei nicht ausschließlich auf Aussagen qualitativer Natur wie z.B. “*Ist das System fehlerfrei?*” Auch lassen sich quantitative Folgerungen treffen. Die Anwendung von Verifikationstechniken im Bereich *Drahtloser Sensornetze* ist Bestandteil dieser Arbeit und gliedert sich in zwei Teile.

Zum Einen wurde untersucht inwieweit sich existierende Modellprüfungsverfahren (Model Checking) zur Ermittlung von quantitativen Energieverbräuchen nutzen lassen. Hierbei wurden geeignete mathematische Modelle basierend auf Timed Automata Theorie entwickelt, in denen sich beispielsweise der Energieverbrauch eines Sensorknotens in einem vorgegebenen Szenario bestimmen lässt. Außerdem können dank der integrierten Mechanismen aus allen erfüllenden Pfaden die einer Suche entsprechen die jeweils energieoptimalen ausgewählt werden.

Für probabilistische Algorithmen existiert dieser Ansatz nur zum Teil, da hier keine erfüllenden Pfade bestimmt werden können. Dennoch lassen sich auch hier durch die Anwendung *Formaler Methoden* entsprechende Aussagen treffen. Diese können zum Beispiel lauten “*mit Wahrscheinlichkeit p läuft das System stabil*”. Bestimmt wurde in der vorliegenden Arbeit die geeignete Wahl der Parameter durch probabilistische Modellprüfung, mit Hilfe derer sich ein kleinst möglicher Energieverbrauch für ein authentifiziertes Transportprotokoll einstellt. Von besonderem Vorteil hat sich die hohe Präzision der Resultate erwiesen da die Ergebnisse berechnet und nicht simuliert werden. Dennoch ließ sich die Methode wegen der schnell steigenden Komplexität der verwendeten Modelle nur auf kleine Sensornetze mit bis zu 20 Knoten anwenden.

Der zweite Teil der Arbeit befasst sich mit der Verifikation von Protokollen und eingebetteter Software aus dem Bereich von Sensornetzen. Die Vorgehensweise ist hierbei häufig die modellbasierte Verifikation. Hierbei wird gemäß der Funktionalität ein System händisch modelliert und später verifiziert.

So wurde gemäß einem Übertragungsprotokoll ein probabilistisches Modell entworfen und gezeigt, dass eine zuvor hergeleitete Formel fehlerhaft ist. Insbesondere wurde bei dem Formalismus — der aus verschiedenen Parametern die Wahrscheinlichkeit berechnet, dass ein Knoten eine gefälschte Anfrage weiterleitet — bewiesen, dass eine Annahme bezüglich der Unabhängigkeit zweier Parameter für die fehlerhafte Wahrscheinlichkeit verantwortlich ist. Im Laufe der Arbeit konnte eine korrekte Formel basierend auf der Entwicklung hyper-geometrischer Reihen hergeleitet und als korrekt bewiesen werden.

Grundsätzlich besteht bei manuell entwickelten Modellen das Problem, dass sich bereits beim Entwurf Mängel einschleichen können, die später im Verlauf der Analyse dann zu unerklärlichen Fehlern und Fehlverhalten führen. Da diese Vorgehensweise fehleranfällig ist und außerdem hinreichend große Modelliererfahrung und -kenntnisse erfordert, ist hier die automatische Modellgenerierung zu empfehlen. Hierbei wird ein Modell entgegen dem zuvor präsentierten Ansatz automatisch abgeleitet.

Als Beispiel hierfür lässt sich die Entwicklungsplattform TinyOS anführen, in der Programmcode für verschiedene Plattformen von Sensorknoten implementiert wird. Mit Hilfe einer geeigneten Abstraktion einer Hardware Plattform lassen sich beliebige Protokolle die in TinyOS existieren in ANSI-C Code überführen und anschließend verifizieren. Da dieser Schritt unter Umständen eine anschließende Anpassung der Softwarequellen erfordert um unter anderem der fehlenden Software-Hardware Kommunikation gerecht zu werden, lässt sich durch geeignete Transformationen der Aufwand zumindest reduzieren was den abschließenden Verifikationsprozess erleichtert.

Auf diese Art und Weise wurde ein *Concast* Protokoll mittels der *NULL* Plattform abgeleitet und untersucht. Als Werkzeug wurde Software Bounded Model Checking angewendet, wodurch sich Korrektheitsaussagen wie typ-korrekte Dereferenzierung von Zeigern, Null-Division und andere Kriterien sich anhand der Software Quellen beweisen lassen. In einem weiteren Schritt wurden Spezifikationen definiert, die beispielsweise die protokoll-konforme Behandlung von Paketen garantieren und im Verlauf verifiziert. Die hier betrachtete Software Verifikation lässt sich jedoch nur auf Programme anwenden, die in TinyOS implementiert wurden.

Desweiteren existieren auch andere Plattformen in denen sich Software für drahtlose Sensornetze entwickeln lässt. Ein Beispiel hierfür sind die SunSPOT, die eine virtuelle Maschine besitzen und die Ausführung eines JAVA Dialekts erlauben. In einer weiteren Untersuchung wurden quell-offene Teile der SunSPOTs Bibliothek auf Korrektheit überprüft. Hierzu wurden Spezifikationen aus verfügbaren Hersteller Dokumenten und auch dem Quellcode erstellt und mit Hilfe eines Theorembeweisers verifiziert.

Abstract

When considering the use of *formal methods*, a still prevailing view is its application of mathematical models and techniques for the analysis of information and communication technologies. Model checking is a branch of this, offering an exhaustive investigation method of the model spanned search space via brute-force algorithms. Hereby the analysis is not only restricted to qualitative statements for example *"Is the system error free?"* Instead, quantitative statements can also be checked. The application of these verification techniques in the area of *wireless sensor networks* is the objective of this work which subdivides in two parts.

In the first part we investigate the extent to which model checking can be used for the analysis of quantitative energy draws. In this sense suitable mathematical models based on timed automata theory are developed which are in turn used for example to determine the energy use of a sensor node in a predefined setting. Furthermore, due to the integrated mechanisms, we can choose among all fulfilling paths that correspond to a search those, which are energy optimal.

For probabilistic systems similar approaches only exist to some degree, since no fulfilling paths are provided by the search algorithm. Nevertheless, similar results can also be obtained in this setting by the application of formal methods. These can for example be *"with probability p the system is running stable"*.

In the work at hand, we determined an appropriate parameter setting for an authenticated dissemination protocol, which delivers a minimal energy draw. In this case the high precision of the obtained results is advantageous, since the outcomes were computed and not simulated. Mainly caused by the fast increasing complexity of the models under investigation, this method can only be applied to relatively small networks of up to 20 nodes.

The second part of this work covers the verification of protocols and embedded software from the field of sensor networks. The procedural method is hereby often based on model based verification. Here, a system is modeled by hand according to a functional description and later on verified.

In this sense, a probabilistic model is designed according to a transport protocol, showing that an earlier derived formula is erroneous. Essentially, the formalism uses different input parameters and computes the probability for a fake query to be forwarded by a legitimate node. Eventually we were able to formally prove that a neglected dependency constraint between two parameters is to blame for the inaccurate probability results. Within the course of this work, a correct formula based on the evolution of the hypergeometric distribution is derived and proved to be correct.

Basically all manually developed systems have the intrinsic problem that within the modeling phase failures can be introduced which are later during the analysis almost impossible to detect. In consequence these errors often lead to inexplicable failures of the system under investigation. Since this approach is error prone and in addition extensive modeling expertise and knowledge is required, the automated model generation is in this sense advisable. Here, a model is derived fully automatically in contrary to the previously presented approach.

As an example we consider the development platform *TinyOS* in use for the implemen-

tation of programs for different sensor node types. By finding a suitable abstraction of a hardware platform, arbitrary programs can be translated from the TinyOS implementation into *ANSI-C* code for the further verification. Under some circumstances, this step requires manual interference to modify the software sources appropriately and adapt them for the missing software-hardware communication. Still, using an adequate translation, the manual overhead is at least reduced, which also facilitates the consecutive verification process.

Doing so, a *Concast* protocol was derived by the use of the *NULL platform* and analyzed. As a tool we chose a software bounded model checker, which is capable to apply correctness checks like type-correct pointer dereferencing, division by null, and other criteria using the software behavior model of the sensor node. In a subsequent step specifications which guarantee a protocol-compliant treatment of packets were inferred. The verification of the specifications afterwards, is an additional element of this work. So far, the considered software verification is only applied to software of sensor nodes implemented in TinyOS.

Moreover, also other platforms exist which allow the development of sensor node specific software. An example for this are the SunSPOT sensor nodes. They run a virtual machine and allow the execution of programs in a JAVA like dialect. We investigated parts of the open-source networking library deployed on the SunSPOTs and checked it for correct behavior. To achieve this, specifications were defined according to available sources like data sheets and by the SunSPOT's source code, and eventually verified using a theorem prover.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Formal Methods

The application of formal methods is still a young technique with increasing popularity, especially in the modern industrial development processes, where software developers of innovative companies recognized its potential. For example companies like Microsoft make great efforts for improving the correctness of code and finding bugs.

Present in academia for more than two decades, this technology has now reached a stage of maturity that allows a wide range of applications starting from abstract models that implement mutual exclusion, up to verification of object oriented software and operating system drivers. For correctness checking of object oriented software there exists a variety of tools like KeY [BHS07], ESC/JAVA [RNS00], or Spec# [BLS04] that gained notability even beyond academic borders. For Verification of C like software tools like BLAST [BHJM07], CBMC [CKL04], VCC [CMST08], HAVOC [CLQR07], and others proved suitability in case studies. Although other analysis techniques exist, the use of formal methods has advantages that are unmatched by any other approach. Especially in areas where secure, safe and dependable systems are crucial, their exhaustive checking in comparison to simulation tools is a unique feature.

This success story of formal methods is mainly due to the fact that simulation does allow for prototypical checking of a system, but they cannot be used to check for all possible system interleavings. In the area of probabilistic algorithms this advantage is a superb feature that distinguishes the application of formal tools from simulation and testing tools. In addition, their mathematical rigor and correctness, allows the precise formulation of the model describing the semantic and the related specifications that a model has to fulfill.

Yet, this technology is still far away from providing push-button proof checks, and in reality the use of these tools is like leading mathematical proofs: It is often a time and work intense business. For checking systems on an abstract level a model needs to be designed with an appropriate phrasing of the desired properties. Essentially all relevant features need modeling and having the proof goals in mind, well suited mechanisms need to be added to the model to allow property checking.

Since this task requires a considerable amount of time and is error prone, approaches exists that ease the modeling and specification part. By automatic model generation a proof-basis can be derived that allows further ongoing analysis. But in many cases the model still needs a manual reconfiguration which requires engineers to have a deep understanding of formal methods and the product under verification. Even though many improvements were introduced that lead the application of formal methods into the industrial application area, much effort still has to be spent on theory, formalisms, algorithms and their final synthesis into usable tools.

1.2 Formal Methods in Wireless Sensor Networks

In the field of networks the use of formal methods is relatively sparse, if present at all and strongly competing with simulation frameworks, although scientists realized the potential of logics and their application to security protocols very early. A prominent formalism in this context is the BAN-logic [BAN90], published in the 1990s that was designed to verify cryptographic aspects of security protocols and has means to account for shared-key, public-key, and hash-functions in signatures. The focus hereby is more on security and cryptography related questions like *"Does the protocol do anything unnecessary that could be left out without weakening it?"* Although logics to verify cryptographic aspects of security protocols were developed, the influence on sensor networks is only minor. In essence, the application of formal methods in the domain of *wireless sensor networks* (*WSN*) suggest itself since programs are relatively simple and the devices are restricted with respect to several points.

To start with, they are small in size in comparison to a desktop computer and the biggest component is their attached energy storage, the battery. Nevertheless, this source of energy can only power a small micro processor running at a few mega hertz and a relatively small memory of at most 1 MB. Since the deployed program and the collected data during runtime share this memory space it is clear, that only simple programs and tasks can be run on the hardware. In case that a program is executed and much information is collected — for example like sensor information or even routing tree information — the admissible program space even decreases further. From the processor speed and the memory restrictions results the fact that only symmetric encryption methods are useful since complex public-key encryption would be too expensive. So far, one could argue that methods which prevailed in the world of personal computers also have a good to establish also in sensor networks world since devices here are much simpler.

A further argument to motivate the use of formal methods is that a sensor network is a collection of devices that only differ in their individual ID or network address. In this sense they are excellent candidates since once a model of a sensor node is designed, it can be copied and adapted using different network addresses or IDs. Under this assumption, such systems can nicely be modeled as a concurrent system, having the additional potential that symmetry reductions can be applied very efficiently.

The question why formal methods are not widely used in wireless sensor networks cannot be answered that easily. One explanation is that in many existing application areas like i.e., structure monitoring, safety is only sometimes considered. As a fact, the deployment area of highly confidential, reliable and dependable systems is in practice less relevant and only a minor issue.

Another reason that the application of formal methods is rarely used in the wireless sensor networking context is that no *off-the-shelf* verification tools exist. Therefore a suitable representation needs to be chosen that often only allows the modeling of some desired aspects. Where in the networking community simulation tools like NS2 [Net] or the TOSSIM emulator [LL03, LLWC03] that comes with the TinyOS platform are widely accepted, there is no such tool that implements formal method verification. Another downside of model checking is that an initial model has to be refined throughout several iteration steps. Reducing a model which allows the application of formal methods is very often a cumbersome and tedious process. Essentially, the complexity has to be reduced by abstracting from irrelevant details in order to emphasize the important aspects and not increase the complexity of the system under investigation. It is even the case that different aspects of a network need diversified modeling and differentiated applications to check them.

As a fact, many arguments can be found that suggest the application of formal techniques in the wireless domain due to the intrinsically distributed character. In addition models can be designed at a refinement level, containing all the features of interest, and abstract from negligible functionality. Once a single sensor node is modeled as a reference design, it can be used to create a wireless network by parallel composition of this model instance. This is a nice property since most of the settings require sensors that have identical algorithms and only their unique ID or networking address make them differentiable.

General areas of interest in the WSN field not covered by simulation tools, hardware testbeds or other testing methods are vast. One of the prominent areas is the verification of sensor node software and it is indisputable that for security related applications they have to fulfill certain safety standards. This application of formal methods to check qualitative aspects is probably the most prominent example and hence mostly known as software verification. Existing tools that check safety and dependable systems must also be applied to software protocols to highlight algorithmic aspects in sensor networks.

In addition, the application of formal methods is not only restricted to the software design process. In particular, quantitative measures can be computed by the concepts like *Timed Automata* [BY04] or *Linearly Priced Timed Automata* [BLR05, RLS04a, LBB$^+$01], a concept that allows the annotation with a function to express costs of a delay or a specific action. By the help of this formalism energy predictions can be stated to forecast for example the maximum life time of a battery powered sensor network. In general, this tool domain can deal with problems related to time, which can also be found in the context of WSN. For example predictions about the *time to response* can be met with an extraordinary high precision thanks to the mathematical foundation.

Another field that gained emphasis is the area of probabilistic algorithms and their analysis. In this context, decisions in a distributed network can be either accomplished by a distributed decision-making through probability or a central entity, sending the decision globally around in the network and hereby causing a communication overhead. So a distributed algorithm can be used as a counter measure avoiding the costly sending and receiving of messages by the use of probabilistic decision-making. The formalisms that allow the modeling of those systems belong to the family of *Markov Processes* and *Markovian Systems*. These methods can be used to analyze reachability of nodes, or energy costs of probabilistic protocols. Here the application of simulation techniques is inadequate since experiments have to be repeated sufficiently often to gain a considerable high precision, and this can sometimes be more than 1 million times. Otherwise confidence criteria are not sufficiently fulfilled and the statistical reasoning about the results can infer errors on the protocol and the overall performance.

1.3 Contributions

This thesis analyzes the applicability of formal methods to the world of wireless sensor nodes and the networks that can be constructed of them. In principle all of the formalisms used are derived from the verification community and proved in previous case studies. Some of the tools are even established and reached a stable and reliable state.

The application of the verification formalisms is twofold. In the first part verification is used to obtain quantitative results about network properties. That is for example, "How much energy can be saved in a scenario when using a specific parameter setting?". The second part treats prominent application of formal methods, that is to check for correctness of software, protocols and algorithms.

1.3.1 Quantitative Energy Estimation

The idea to use methods from the field of formal verification and apply them for constraint solving in the context of wireless sensor networks was already proposed in case studies before. In addition case studies exists that analyze performance issues in combination with general safety and liveness properties of security protocols or network protocols in general like the IEEE 1394 Firewire standard. In the work of [TCCP08] security aspects of the μ-Tesla protocol are analyzed. The work of [FvHM07a] models and verifies the LMAC protocol, a medium access control protocol for wireless sensor networks. In their approach they investigate all possible connected topologies consisting of four up to five nodes and focus in mainly on the detection and resolution of collisions.

The innovation with the approach as shown in this thesis is to apply techniques known from the verification domain to problems in the sensor network domain to efficiently compute energy usage. As known to the author, there exists no comparable work that incorporates the energy aspect into models. The possibilities that show up in combination with the versatile propositional logic offer a variety of investigation possibilities. As such any thinkable scenario can be easily and quickly configured. Thanks to the exhaustive character of the underlying model checking framework, any situation that can occur, can be computed. We consider situations that only show up with a vanishing but still considerable probability, and precisely compute the energy spent in these settings. Essentially these situations are even hard to detect in simulations due to their rare occurrence.

Results furthermore indicate that the Timed Automata formalism is an appropriate candidate for modeling a sensor network. As it turns out, the successful analysis of a network is only promising if a notion of time is present that can be incorporated into the specification. Moreover the use of non-determinism makes the model very flexible, and offers a compact design and human readable models.

Probabilistic decision-making is a widely used method in distributed systems since no additional message passing is required. By running the algorithm distributed entities can option autonomously. Since Timed Automata simply have no mean to express probabilistic branching, a probabilistic modeling formalism is required that is capable of handling probabilities. Such a formalism is for example represented by the class of *Markovian processes* and *chains*, that offer constructs to account for likelihood, but in turn lack a notion of time.

For probabilistic systems, studies exists that investigate for example dynamic power management switching [NPK$^+$02, NPK$^+$03] where based on a randomized power management strategy the average power dissipation should be minimized. In the context of energy optimizing is also the work of [KNP05] that investigated the performance of real-time embedded systems using dynamic voltage scaling. Low-rate wireless personal area networks are explored using probabilistic timed automata theory in [Fru06]. The tradeoff hereby is that either the battery life is prolonged or the performance of the battery powered processor in increased.

In the present work we drop the time notion since the probabilistic effects have a stronger impact on the results. A model based on Markov chains is developed and applied to an existing secure query dissemination protocol. As in the analysis of the previous chapter the focus of this analysis is on energy related questions. Based on the results it is possible to claim that different topologies are more or less vulnerable to an attack of the investigated probabilistic protocol. Further on, it turns out when using the probabilistic query dissemination scheme that nodes close to the adversary are more vulnerable to forged queries and more concerned about energy draining of forged queries.

Essentially it shows how to select the protocol parameters in an optimal way to obtain maximum security at a minimum energy use. In this way, a cost optimal setting is derived, depending on the estimated frequency on which attacks happen.

1.3.2 Protocol and Software Correctness

The second part of this work applies general verification methods in the general sense, namely to verify the correctness of algorithms. It is indeed the case that each algorithm needs to be considered on its own and very seldom there is a beneficial contribution from related work. Although literature is full of correctness proofs for all kinds of protocols, they strongly depend on the context.

In this thesis a networking protocol for authenticated query dissemination (simple authenticated query flooding – sAQF) is investigated. To the best of our knowledge there is no such correctness proof for the protocol under investigation since the verification was applied during the design of the algorithm and even before its publication.

Since the proposed protocol was new and no formal investigation existed, the approach required special modeling of the analyzed topic. It shows that during the formal evaluation of the Markov model, errors in a simulation script, that was developed independently from the formal analysis, were found. As the simulation results would end in wrong assumptions and a bogus choice of parameters, the overall efficiency would be not correctly computed and any conclusion drawn would be simply wrong. It is considered as a fact, that without the help of the formal investigation this modeling error would have never been found, and having a simulation and a formal model both strengthen the belief in the implemented models.

Further on, for the evaluation of the parameters and the derived probabilistic formula, the presented analysis reveals a neglected dependability constraint. In the course of this work, a correct probability formula is derived that is formally verified to compute the correct probability. The use of rigorous techniques applied during the analysis of the authenticated query flooding protocol helped in the present case to find potential errors that have a tremendous impact on the performance of the protocol.

The fact that the model was artificially created does not enforce the belief in its correctness, although eventually the simulation and the corrected formula suggest identical results. And for more complex software systems the model based verification approach is far from being feasible. In addition, errors can be introduced into the model that hamper the verification process.

The automated application of correctness checks for TinyOS programs is studied in the work of [KMG08] using a tool called *FSMGen*. In addition the *Slede* tool [Han07] offers a possibility to investigate protocols for a fixed topology using a protocol model generator.

In this work the approach is different in the following sense. Instead of deriving an abstraction of the model, a fully functional behavior model is generated and analyzed which contains a complete low-level description of the sensor node in a software model. This has the advantage, that no model is needed, since the original sources of the protocol implementation are considered. Secondly, the user must not learn another formalism to understand the details of the implementation since they are expressed in ANSI-C. And in the end, very powerful tools already exist in the community for the verification of software. Using the proposed approach we benefit essentially from synergistic effects. To demonstrate the capabilities of the recommended analysis, we exemplify it by the analysis of a concast protocol. For the verification of properties we use the technique of software bounded model checking, employed in a tool that in addition offers general correctness checks applicable to software code, like type-correct pointer dereferences, division by zero, etc. Since the distributed behavior and the concurrent character of sensor networks are difficult to investigate, a distributed scenario is designed which is analyzed by the use of conventional model checking.

Besides the TinyOS framework which represents the classical embedded system approach also other engineering platforms emerged lately. An example for a JAVA-based development platform are the SunSPOTs which run a virtual machine. In this thesis we survey the SunSPOT's networking library. Since the library consists of more than ten thousand lines of code, we restrict the analysis to a small part which considered the LoWPAN layer (low-power wireless personal area network). One functional aspect is verified in this work that covers the dispatch-mechanism. In this investigation specifications are derived from product data sheets and network standards and verified by the use of a theorem prover called KeY. As shown by the case study, state-of-the-art tools exist which are extremely powerful, but they have evident limits. In the present example low-level operations like bit-vector handling is still very complicated matter even for the use of sophisticated software and potent computing hardware.

1.4 Thesis Outline

The thesis is organized as follows. In the next Chapter 2 the two fundamental views which are comprised in this thesis are introduced. This is on the one hand the concept of *Formal Method* that also subsume the application of model checking. And on the other hand the ideas and technical details from the field of *wireless sensor networks*, indicating their applicability, and weakness. In addition, this chapter gives an overview about the logics, theory of automata, and their implementation in tools.

After this introduction and foundation, the first of the two parts of the thesis starts which covers the analysis of energy consumption and how they can efficiently be computed using different theoretical concepts. In Chapter 3 a sensor network scenario is analyzed based on the theoretic model of *timed automata* with respect to safety, and liveness properties, and in addition energy draws are added to investigate the energy spent in different settings. Chapter 4 analyzes the energy use of a probabilistic protocol for authenticated query dissemination called *AQF* (*authenticated query flooding*) in dependency of an adverse entity.

The second part covers qualitative aspects of software verification like correctness of protocols. A query dissemination protocol for particular small networks called *simple authenticated query flooding* (*sAQF*) is checked for correctness using probabilistic models in Chapter 5. Thereafter Chapter 6 investigates the correctness of the *Concast* protocol *ESAWN* (*Extended Secure Aggregation for Wireless Sensor Networks*) by means of the SPIN model checker and software bounded model checking (SBMC). In Chapter 7 parts of the SunSPOT's networking library in charge for the sending of packets is verified using the KeY tool.

The rest of the thesis is understood as a closure: In the final Chapter 8 the present work is wrapped up, giving a short summary about the presented results and further tasks for future work are proposed that result from the contribution of this thesis.

CHAPTER 2

Foundation

2.1 Introduction

This chapter is a foundational part of this work since it gives a general overview about the topic of this thesis. It furthermore introduces the key techniques from the area of formal methods and some of the tools which made it beyond the borders of academia. Since their understanding is relevant for the comprehension of the whole subject, they are explained in detail, and additional references for further reading are provided where needed.

Also in the world of the sensor nodes conventions must be explained to establish a common view on devices. The way nodes are built, what their key features are, their drawbacks, but also their strength are essential for the understanding of the following chapters. For the definition of safety and security related properties, an appropriate threat model is introduced to better understand the covered aspects of protocols, their weakness and points of intrusion. For this reason this part lays the principal building blocks relevant for the understanding of the following analysis where this adversary is modeled and its abilities are considered when arguing about protocol safety.

The field of formal methods as well as the area of sensor networks are still very active fields of research. This does not only hold for the presented applications that rapidly change. Also their efficiency and the implemented algorithms are continuously improved. It is also valid for the wireless node world where devices become smaller and smaller and their effectiveness steadily increases. In this sense, it is very likely that in a few years from now some of the presented material will be subject to changes but the techniques in principle will remain.

This chapter bridges the gap between the theoretical formalisms in the model checking world and the application area of sensor node devices which is a central part of this work. It is relevant since major aspect of this thesis investigates the usability of proof techniques in the area of sensor devices or more accurately *"Applied Formal Methods in Wireless Sensor Networks"*.

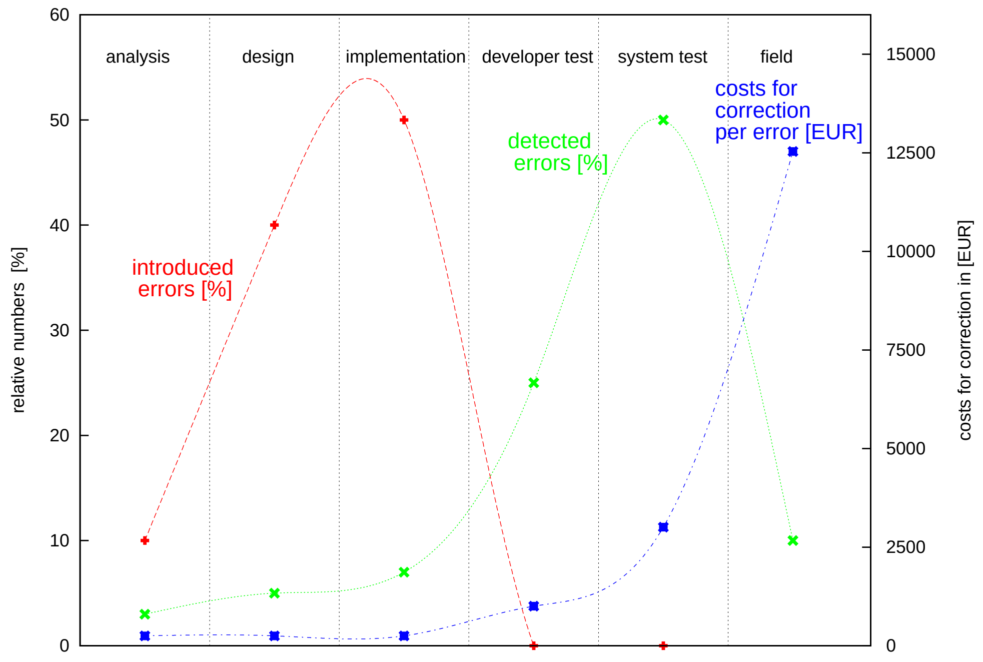


Figure 2.1: Empirical Data related to errors and costs (see [Möl96]).

Overview This foundation chapter is structured in the following way. In Section 2.2 fundamental aspects of this work are mentioned, covering the application of formal methods in general, and the use of model checking techniques. In the subsequent Section 2.3 we present tools used for the verification task. Along with their description we provide a short abstract about implemented theories, and give references to the literature. In the last section (see 2.4) the major aspects of wireless sensor networks are explained including an adversary model, and different types of sensor node platforms.

2.2 Formal Verification Techniques

Since the design of software and hardware systems is increasing in complexity, more time is spent on checking their correctness than on their actual construction [Kat03]. The demand for appropriate techniques that increase the coverage of verification efforts while reducing their work load is present. As case studies showed about 90% of all errors are introduced within the early design and implementation phase of a product (see Figure 2.1). In addition, detecting such defects at an early state is relatively cheap, but if errors stay undetected up to the system test or even the field test, it is causing massive expenditures to fix them. By applying formal methods in the product development cycle and integrating them actively into the design process the required time and the costs for verification are reduced.

What is in fact hidden behind the term *formal methods* is a sound mathematical structure that allows for modeling and the later analysis of systems mainly from the information and communication domain. And through their mathematical rigor, the correctness of highly dependable systems can be established. In the fields of software and hardware verification of safety-critical systems, the use of formal methods is widely adopted due to its potential. During the last decade there has been enormous success in the development of verification techniques that lead to sophisticated tools and various automation steps in the verification process. In fact by the use of formal procedures, investigations have revealed prominent defects as in systems, like the Ariane-5 Missile [Mar97], Mars Pathfinder, or the Intel's Pentium II bug.

In general two distinct fields of formal verification exist. *Deductive methods* belong to the one area and can be found in tools like theorem provers and proof checkers that determine the correctness of a system through a mathematical theory with the highest possible precision. On the other side *model-based verification* exists, describing the system through mathematical models in a precise and unambiguous fashion. The verification is accomplished by using a system model that consists of an accurate definition and a specification, and all possible modeling states are exhaustively explored with the help of a search algorithm. Many specialized verification techniques exists that range from experiments in reality known as *testing*, *simulations* executed in a restrictive set of scenarios, up to the exhaustive exploration of all possible states, referred as *model checking*.

In this work the focus in on the later technique namely *model checking* although the others do play an important role and thus their major aspects are just briefly introduced. For an in-depth explanation refer to the work of [Kat03].

2.2.1 Model-based Simulation

When deciding about the quality of a prototypical implementation, *model-based simulation* is in particular useful. To achieve this, a model obtains external stimuli and the observable reaction of the model can help to reveal early design flaws in the implementation. This can never be an exhaustive checking since for most systems it is infeasible to analyze all possible system settings by simulation. Due to this disadvantage generating exhaustive scenarios for the simulation is time consuming and very costly. And in fact unexplored scenarios may still remain that can contain a flaw. Furthermore assertion about the systems degree of correctness — the so called *coverage criteria* — are hard to quantify and estimations are only very rough. For many scenarios a safety guarantee stating that *in 99 percent of all cases, the system is fine* is sufficient, but due to the afore mentioned problems even these statements can hardly be proposed.

2.2.2 Model-based Testing

The application of *model-based testing* is useful in areas were the creation of a system model is hard or even impossible to obtain. Depending on the accessibility of the system internals, different kinds of testing are feasible. So called *white box testing* is applicable where the internal structure of the system is fully accessible. If only parts of the internals are available to build tests, *gray box testing* is accomplished, and last *black box testing* is employed if there is no access to internal details and their structure is completely hidden. One of the major advantages of testing is their application to final products as opposing to restricted models that only represent a subset of the fully implemented features. But similar to simulation techniques as mentioned before, model based "testing can only show the presence of errors, never their absence" [Dij72].

2.2.3 Theorem Proving

Theorem proving is using mathematical theory in terms of a logic to infer the system's correctness. So the system needs to be transformed into some logic. In combination with a set of axioms — that are fundamental laws formulated as theorems which are always valid — a proof is generated by the theorem prover.

The major advantage of proof checkers is their applicability to infinite state systems since they rely on mathematical principles such as *structural induction*. Since they make use of parameter settings as non-discrete values, their application is not restricted to numerical application scenarios. Although successful applications of theorem proving exist, developing these proofs is usually very time consuming, labor intensive and prone to errors.

The implemented logics by theorem provers range from variants of first order logics, typed logics, up to higher logics (typed-order logics where variables range over function types of predicate types) that can handle object oriented programming languages. Prominent proof checkers for higher-order logics are *PVS* [SOR93], *Coq* [BC04], *Isabelle* [Pau94] etc. For the verification of software a group of theorem provers are well-established that transform a program into an appropriate logical representation. Tools that belong to this category of theorem provers are the KeY [BHS07], or BLAST [BHJM07].

The last prominent technique in the field of model based verification is model checking which is introduced in full detail in the following.

2.2.4 Model Checking

The term *Model Checking* was coined in the early eighties by the work of Clarke and Emerson [CE82] who worked on automatic brute force examination of programs automatically synthesized from a high-level temporal logic specification. Independently from the above Queille and Sifakis [QS82] discovered similar methods when analyzing concurrent systems in CESAR.

The technique behind model checking explores all possible system states of a model $\mathcal{M}$ under investigation through efficient algorithms and data structures in a systematic manner. Hence if a state satisfying a property φ is found, a mathematical proof is provided that can be used to compute the conditions that lead to it. The challenge is to examine the largest possible state space that still fits into the memory and which can be searched for property φ. The maximum size of the state space varies for state-of-the-art model checkers between 10^8 and 10^9 states with explicit state-space enumeration. Using sufficiently sophisticated algorithms and data structures, larger state-spaces consisting of up to 10^{476} states [Kat03] lie still within the feasibility for some problem instances. And in consequence even subtle errors that remain undiscovered by emulation, testing and simulation can potentially be revealed by the model checking approach.

The majority of checkable problems have a qualitative nature like "Does the system ever recover from a failure?" or "Can a deadlock situation be reached?" The term *deadlock* refers hereby to an unintended halt of the entire system mostly caused by concurrent processes mutually waiting for each other. Also *livelocks* are possible where processes cycle continuously between different states forever, but cannot escape this sequence.

The system model is often automatically generated from model descriptions specified in some appropriate programming language like *C*, JAVA, or hardware description languages such as *Verilog* or *VHDL*. But it can also be the case that it is manually created. The model description *How does the system behave?* and the property specification *What should the system do and what not?* are then fed into the model checker which examines all possible states satisfying the property. If a violating state is found, a counter-example is provided showing the path from the initial state to the violating system configuration.

The technique of model checking as been applied to many fields ranging from the application of on-line airline reservation systems, over modern e-commerce protocols up to software employed in the storm surge barrier in the harbor of Rotterdam [CTW99, Pim98]. During these investigations several serious design issues were revealed that did not show during testing and simulation.

In the area of wireless sensor networks the application of formal methods has recently started. Since the method is very flexible and versatile, it is applicable to all systems with the appropriate degree of realism. For distributed systems — and WSN can be as such considered as a distributed system — their application is extremely suited through models like finite state machines that especially emphasize their concurrent nature. Fortunately, in most of the application domains in the field of wireless sensor networks one is interested in only a small set of properties requiring only a small fraction of the temporal logic. In fact, this matter allows the user a degree of freedom to choose the appropriate modeling mechanism which best reflects the model under consideration.

But there are also downsides involved with the model checking approach that one should keep in mind. The probably most prevailing to name addresses the size of the system, since only finite models can be considered with the proposed technique. Otherwise the verification algorithm is running out of space and does not provide any answer about the property of interest. This problem is intrinsic for all tools that employ the model checking technique and is referred in literature as the *state space explosion* problem.

Over the years many technologies to combat the downsides of model checking have been implemented. In principle they all introduce counter measures for the fast and accelerating increase in the model size and the resulting state space.

A commonly used method is to use abstraction and hereby decrease the complexity to an extent which makes the application of model checking feasible. Since right at the beginning one is interested in predefined properties, the simplification can reduce unnecessary details that do not influence the considered behavior. Nevertheless, the reduced system can only be checked if the applied abstraction is sound, meaning that the properties on the original model must also hold on the reduced model. Since the reduction of the model freed resources due to a decrease in the model's complexity, subsequent refinement processes are possible where features of interest can be modeled now in detail.

For explicit state model checking tools the application of *partial order reduction* can lead to a tremendous decrease of the model's complexity by a magnitude of more than hundred. It works on the generated explicit state graph by finding independent interleaving of concurrent processes that can be safely removed from the model. In essence, for many parallel and similarly behaving processes as found in the domain of wireless sensor networks, the complexity savings are mostly due to symmetry reduction techniques.

By the use of symbolic algorithms the performance of model checking can be increased as well. With this technique no graph of the finite state machine is built up in memory. In contrary, a graph implicit representation for the propositional formula is chosen.

2.3 Tools and Theory

The tools presented in this section implement completely or partially different theories. For this reason we give a short overview of the software used for the modeling and verification, and introduce their theoretical background where required.

2.3.1 SPIN

The development of the SPIN *model checker* [Spi09] started 1980 at the Computing Sciences Research Center of Bell Labs. Since 1991 the software is freely available. It fully implements the *linear temporal logic* (*LTL*) that offers modalities to reason about time, like the future of paths under which a condition should evaluate to true. The syntax of the propositional logic is given by the following definition.

Definition 1 (Linear Temporal Logic – Syntax) *For an atomic proposition ap, and valid LTL formulas Φ, Ψ, formulas in LTL are constructed according to the following rules:*

$$\Phi ::= ap \quad | \quad \neg\Phi \quad | \quad \Phi \vee \Psi \quad | \quad \mathcal{X}\Phi \quad | \quad \Phi\,\mathcal{U}\,\Psi \quad | \quad \Box\Phi \quad | \quad \Diamond\Phi$$

The semantics of LTL is defined by quantification over a single path. Hereby ap is a propositional formula which only depends on the current state. Further on, $\neg\Phi$ is satisfied, if Φ does not hold on the path, and $\Phi \vee \Psi$ is true if either Φ or Ψ holds on the path. The claim $\mathcal{X}\,\Phi$ is satisfied, if Φ is satisfied in the next state, meaning the actual state plays no role. The construct $\Phi\,\mathcal{U}\,\Psi$ states that Ψ has to hold in a successor state, and in all of the previous states Φ has to hold. The last two LTL formulae are auxiliary and do not add expressiveness to the language. In fact, they facilitate the process of specifying complex properties and in spoken language they read as *eventually* and *always* properties.

SPIN models are defined in *Promela*, a *process meta language* that is further explained in [Hol03]. Promela models consist of asynchronous processes defined by `proctype`, synchronization statements, and message channels. The final automata is instantiated using the defined processes with interleaving. Inter-process communication is realized using channels `chans` that can be either buffered with an arbitrary size or unbuffered. Preprocessor commands can be added to Promela which appear at the very beginning of a model definition similar to C. In addition complex data types can be defined. Atomicity can be arbitrary added to enforce atomic block building by the construct `atomic{..}`. By the use of keywords `xr` and `xs` an exclusive read- and write access — so called *channel assertion* — is set on channels which is checked by the tool at runtime.

SPIN is able to deal with huge state spaces. In particular, by use of the implemented partial order reduction techniques and on-the-fly interpretation, that avoids the construction of an explicit Kripke structure, efficient model checking is possible. Furthermore C-Code can be embedded in the Promela model to act as a real implementation.

The SPIN model checker offers different modes of operation. It can act as an interactive simulator which is helpful in analyzing the model for sanity, or the exhaustive verifier can check arbitrary LTL formulas for validity.

formula	natural language	type
$\Box p$	always p	invariance
$\Diamond p$	eventually p	guarantee
$p \rightarrow \Diamond q$	p implies eventually q	response
$p \rightarrow q \mathcal{U} r$	p implies q until r	precedence
$\Box\Diamond p$	always eventually p	recurrence (progress)
$\Diamond\Box p$	eventually always p	stability (non-progress)
$\Diamond p \rightarrow \Diamond q$	eventually p implies eventually q	correlation

Table 2.1: Overview of often used LTL formulae with their meaning.

Basic properties that can be analyzed through SPIN are properties like safety checks in terms of *validity of end states* and assertions. Over paths properties declarations like *non-progress cycle* and *acceptance cycle* existence are possible which are known as *liveness checks*. Furthermore SPIN allows to check for *never claims*, correctness of exclusive access on variables and channels, and the existence of unreachable code fragments.

The syntax from LTL can be directly translated into SPIN syntax. In this sense operator `!` is for negation, || for disjunction, `U` as the until operator, and `X` as the next operator. The resulting SPIN syntax for property specification in LTL is:

$$\begin{array}{rcl} \langle\text{ atomic proposition }\rangle & ::= & \texttt{true} \quad | \quad \texttt{false} \quad | \quad \langle\text{ boolean expression }\rangle \\ \langle\text{ path formula }\rangle & ::= & \langle\text{ atomic proposition }\rangle \quad | \quad \texttt{X}\langle\text{ path formula }\rangle \quad | \\ & & \texttt{!}\langle\text{ path formula }\rangle \quad | \\ & & \langle\text{ path formula }\rangle \,\|\, \langle\text{ path formula }\rangle \quad | \\ & & \langle\text{ path formula }\rangle \;\texttt{U}\; \langle\text{ path formula }\rangle \end{array}$$

There exist a number of linear temporal properties often used to state assertions about recurrence or response properties. The most prominent LTL statements are listed in Table 2.1.

2.3.2 Uppaal

UPPAAL [BDL04, LPY97] is based on the theory of *Timed Safety Automata* [HCSY92, HNSY94], that can be used for the verification of real-time systems. In UPPAAL a model is a representation of automata $\mathcal{A}$ that is extended with a *clock* variable. All clocks run synchronously and have the same notion of progress. If one clock is advancing, all the other clocks in the system need to advance by the same portion. Clocks are initially set to zero, but then *tick* as automata transitions are executed. In addition transitions are executed instantaneously, without the elapse of time. Constraints and guards on clocks and invariants are defined in [Kat03]:

Definition 2 (Clock Constraints) *Let $\mathcal{C}$ be a set of clocks, $x \in \mathcal{C}$, and c a natural number. The clock constraints over $\mathcal{C}$ denoted as $Constraints_{\mathcal{C}}$ satisfy the following rules:*

- *$x < c$ and $x \leq c$ are clock constraints*
- *if α is a clock constraints, then $\neg\alpha$ is a clock constraint*

- *if α and β are clock constraints, the $\alpha \wedge \beta$ is a clock constraint*

Important to note is that clock constraints need to follow a very restrictive syntax, because otherwise the model checking problem becomes undecidable. So for clocks α, β and $a \in \mathbb{R}^+, b \in \mathbb{N}$ the term $\alpha \leq a + b$ is admissible whereas two clocks on the left side like $\alpha + \beta \leq a$ are not. Furthermore, rational clocks constraints are valid since they can be transformed into natural numbers by scaling all clocks of the system by the least common multiple of the denominators of all constants that appear in conditions or invariants.

Automata can synchronize on channels by the use of *binary-* or *broadcast* synchronization. For each synchronization tuple `C`, a sender `C!` and a receiver `C?` are explicitly defined by the user. The difference between binary and broadcasting lies is the following. In a binary synchronization a sender and a receiver have to be present, otherwise the system's sender is blocked. In the broadcasting scheme the sender is never blocked and instead of having one receiver, there can be multiple ones at the same instance.

State changes within an automaton occur by taking transitions. Like the following transition $s \overset{\mathtt{g,c,u}}{\rightarrow} s'$ from state s to s' ($s, s' \in S$), each transition comprises the following three elements of which all are optional. An edge is enabled if the guard `g` — a construct on the clocks and integer variables — evaluates to true. In case that a synchronization partner is present for a channel `c`, transitions are jointly executed. Afterwards, the update `u` is updating variables and executes the clock resets. If no guards or synchronization terms are present, the edge is enabled.

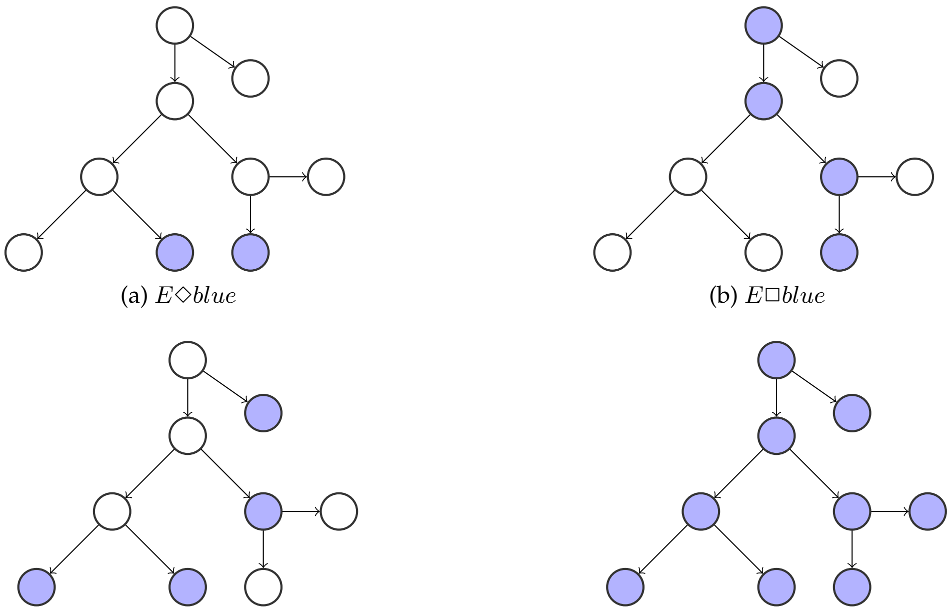
Essentially two kinds of time constraints on the automata exist. On the one hand clock variables in the form of guards are possible on transitions with the notion that an edge is only enabled if the guard is fulfilled. Or they can be added as an invariant to states, that allow an automaton to stay in this state as long as the invariant is not violated. Hence a state must be left in order not to violate the invariant. Invariants can be understood as the complement to guards in the sense that guards allow a state to be left and location invariants enforce such state change.

In UPPAAL location labels `urgent` and `committed` exist, that can be understood as invariants over time on states. An `urgent` label indicates that no time is allowed to pass while the system is in this state. For `committed` locations it is even the case that on the next step each `committed` location must be left.

Each automaton is included in the system through a template that allows to instantiate arbitrary combinations of modeled automata. Since modeling in UPPAAL is mostly done in the graphical interface, the definition of the syntax is omitted here, but can be found in [BDL04].

The Semantics of an UPPAAL system automaton can be explained using the concepts of *action* and *timed trace*. The automaton concept from [HCSY92, HNSY94] comprises of a set Σ of actions. For UPPAAL automata this is a derived concept.

The behavior of an UPPAAL automaton $\mathcal{A}$ is defined by the set $run(\mathcal{A})$ of all its possible *runs*. A run is a finite or infinite sequence $s_0, s_1, \ldots, s_i \ldots$ of *states*. A state in turn is a pair (ℓ, u) of a location $\ell \in L$, and a function u that associates values to all variables and clocks. Of course, we require that ℓ_0 is the initial state and u_0 assigns the initial values. If we look at a system of automata the state of the system is the pair $(\vec{\ell}, u)$ where $\vec{\ell}$ is a vector $(\ell^1, \ldots, \ell^k)$ of locations for all automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ in the system, and u assigns



(a) $E\Diamond blue$

(b) $E\Box blue$

(c) $A\Diamond blue$

(d) $A\Box blue$

Figure 2.2: Valid states for some basic CTL properties are marked by blue nodes.

values to all local-, and global variables, and clocks. A sequence $s_0, s_1, \dots, s_i \dots$ of states is in $run(\mathcal{A})$ if there is a timed trace $(t_1, b_1), \dots, (t_i, b_i), \dots$ that demonstrates that $\mathcal{A}$ can reach the given states in the given order. This is to say that for all i the automaton $\mathcal{A}$ can change from state $s_{i-1} = (\ell_{i-1}, u_{i-1})$ to $s_i = (\ell_i, u_i)$ via the timed action (t_i, b_i). This change comes in two parts. In the first part from (ℓ_{i-1}, u_{i-1}) to (ℓ_{i-1}, u'_{i-1}) only the clock variables change by $d_i = t_i - t_{i-1}$ (with $t_0 = 0$), i.e., u and u' differ only on variables $x \in \mathcal{C}$, and $u'_{i-1}(x) = u_{i-1}(x) + d_i$. The second part is the firing of the edge or the pair of edges b_i in state (ℓ_{i-1}, u'_{i-1}) with end states $s_i = (\ell_i, u_i)$ as already explained above.

A CTL like Query Language is used to specify properties that allow not only to make assertions about one possible successor state [Kat03], but over a whole branch of the following states. As in CTL, the query language allows assertions like *some* or *all* of the states starting in a starting state s fulfill a property, something which cannot be expressed through a LTL formula. In general, CTL distinguishes between state formulae that allow quantification over states, and path formulae expressing the property of a path. Essentially the UPPAAL query language does only handle a simple subset of CTL without the nesting of properties. Some basic properties are shown in Figure 2.2.

2.3.3 Prism

PRISM [Pri] is a *probabilistic model checker* for modeling and the analysis of probabilistic systems. Three types of probabilistic models are supported, which are *discrete-time Markov chain* (*DTMCs*), *Markov decision process* (*MDP*), and *continuous-time Markov chains* (*CTMCs*). Since CTMCs are not used within this work, we refrain from further explaining its syntax and semantics. PRISM internally uses *multi-terminal binary decision diagrams* (*MTBDDs*)

PRISM coded	composition	involved actions
`mod_1 || mod_2`	full parallel	all
`mod_1 ||| mod_2`	asynchronous parallel	none
`mod_1 |[act_1,..,act_n]| mod_2`	restricted parallel	only `act_1..act_n`
`mod_1 {act_1,..,act_n}`	action hiding	none
`mod_1 {act_1<-act_2,..}`	action renaming	`act_2`

Table 2.2: Parallel composition constructs in PRISM.

which extend BDDs through the representation of arbitrary function ranges instead of values 0 or 1.

The PRISM language is state-based and derived from the formalism of reactive modules [AH99]. Models in the PRISM context are composed of modules through parallel composition where variable hiding etc. is possible. Each module has local variables that constitute the state of the module. The behavior of a module is described through commands of the form:

```
[action] guard -> p_1:update_1 + ... + p_n:update_n
```

In the above command `guard` is a predicate over all module variables, and if it evaluates to true, the operations next to the arrow are executed. `p_1` up to `p_n` describe the probability with which their related update is executed. What is important to note is that in DTMC models, the sum of all `p_`*i*'s has to be equal one where is a MDP modeling this does not need to hold. `update_`*i* describes the updates that are executed if the guard is fulfilled. Via the action label `[action]` which can even be the empty action label `[]` concurrent modules can synchronize their commands.

Variables can be of type *Boolean* or *integer* or even define a range of integers. In addition it is possible to assign variables with an initial value using the keyword `initial`. The composition of modules is accomplished by the use of the operators shown in Table 2.2. If no explicit composition is given, full parallel composition is assumed for all modules. Global variables are prefixed with the key word `global` and appear before the module declaration.

Costs and Rewards can be added to model a wider range of quantitative measures. For example questions like "what is the expected time until failure" or "what is the expected power consumption?" are feasible through this. Rewards are added to the model using a `rewards ... endrewards` construct. Within this construct different rewards are added using a `guard:reward` scheme, meaning that whenever the `guard` is fulfilled, the value of `reward` is accumulated to express the expected costs.

Property Specification is accomplished using a querying language based on the temporal logics of Probabilistic Computation Tree Logic (PCTL) [HJ94, BdA95] for DTMCs/MDPs, and Continuous Stochastic Logic (CSL) [ASSB96, BKH99] for CTMC models.

Specifications for DTMCs and Markov Processes are defined using *PCTL* (*Probabilistic Computation Tree Logic*), which is a probabilistic extension of CTL [RKNP04]. Like in CTL, the user is not only able to make assertions about one possible successor state [Kat03], but also over a whole branch of the following states. PCTL formulae are interpreted over states of Markov chains. We distinguish between state formulae Φ, and path formulae ψ that are evaluated over states and respectively over paths. When specifying a property, one starts with a state formula since path formulae only occur as a parameter of the $\mathcal{P}_{\bowtie p}[\psi]$ operator. In other words, a state s satisfies $\mathcal{P}_{\bowtie p}[\psi]$ if the probability of taking a path starting at s satisfying ψ is in the interval $\bowtie p$ with $\bowtie \in \{\leq, <, \geq, >\}$, and $p \in [0, 1]$.

A cost or reward operator $\mathcal{E}$ is added to the language specification. As opposed to model checking of timed automata in PCTL it is not possible to determine the probability with which a certain path formula is satisfied and thus no failure states of the model can be identified. It can only be determined whether the computed probability is within a certain bound. The PCTL syntax is given through

$$\begin{array}{llllllll} \Phi ::= & \texttt{true} & \mid & \texttt{false} & \mid & \langle expression \rangle & \mid & \\ & \Phi \;\&\; \Phi' & \mid & \Phi \mid \Phi' & \mid & \texttt{P bound [} \Psi \texttt{]} & \mid & \texttt{S bound [} \Phi \texttt{]} \end{array}$$

with `bound` equal the $\bowtie p$ operator and $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0, 1]$, as mentioned before.

$$\begin{array}{llllllll} \psi ::= & \texttt{X}\,\Phi & \mid & \Phi\,\texttt{U}\,\Phi' & \mid & \Phi\,\texttt{U time}\,\Phi' & \mid & \\ & \texttt{F}\,\Phi & \mid & \texttt{F time}\,\Phi & \mid & \texttt{G}\,\Phi & \mid & \texttt{G time}\,\Phi \end{array}$$

and

```
time::= >=t | <=t | [t,t].
```

The operator X represents the next operation, `U` the until, `F` the *eventually* and `G` has the notion of *always* as known from other logics.

Reward based Properties can be added to PRISM models to analyze properties that relate to the expected value of certain reward variables. A reward based operator is denoted i.e., as `R bound [ rewardprop ]`. The following four reward properties are supported in PRISM:

The *reachability reward* written as `F prop` refers to the accumulated reward along the path until the specified point is reached. For DTMCs and MDPs the total reward can be counted as the total sum of all state rewards assigned along the path plus the sum of transition rewards. The *cumulative reward* property written as `C<=t` associates the model's reward of all paths up to the time bound `t`. *Instantaneous rewards* denoted by `I=t` refer to a reward at a certain instance of time `t`. And last, *steady-state reward* properties can only be applied for CTMCs and compute the rewards in the long run, when the steady state is reached.

PCTL Model Checking The way *PCTL model checking* works is explained in detail in [RKNP04]: Let $\mathcal{D} = (S, s_0, \mathbf{P}, label, \mathbf{C})$ be a labeled DTMC with S a finite set of states, $s_0 \in S$ the initial state, $\mathbf{P} : S \times S \rightarrow [0, 1]$ the transition matrix, *label* a labeling function, and $\mathcal{C}$ a cost function. As input to the algorithm DTMC $\mathcal{D}$ and a PCTL formula Φ is taken. The output $Sat(\Phi) = \{s \in S | \models \Phi\}$ that is generated and contains all model states satisfying Φ. In the parse tree that is constructed according to formula Φ, each node is labeled with a sub formula of Φ and leaves are labeled with either $true$ or an atomic proposition. From the leaves upwards to the root of the tree the set of satisfying states is computed for each sub formula contained. Upon termination the algorithm determined whether each state satisfies formula Φ.

2.3.4 C Bounded Model Checker

CBMC (*C bounded model checker*) [CKL04] is a tool that can be used to verify low level ANSI-C code. It can handle almost all ANSI-C language features like recursion, float and double types, dynamic memory allocation, and pointer arithmetics. In addition to user specified properties, the tool can automatically derive properties that enforce code safety through pointer checks, array bounds, division by zero checks, and unwinding assertions to ensure that the user defined bound is sufficiently large. The user can hereby add assertions into the C source code and check them.

In bounded model checking, a transition relation with its specification is jointly unrolled up to a user defined upper bound. The resulting Boolean formula can afterwards be checked for satisfiability. If it is satisfiable, an error trace leading to that error will be generated using the SAT procedure. Since unwinding assertions are automatically added by the tool, it is ensured that enough unwinding are accomplished. The transformation is described in full detail in [CKY03] and occurs in five steps.

Within these steps the ANSI-C program is brought into a bit vector equation by first removing side effects that come from `break` , `continue` and `for` statements. Afterwards only while loops are present which are unrolled n times and appended to an `if`-guard carrying the same condition as the original while loop. The added *unwinding assertion* assures that sufficient loop unwinding are done and n is sufficiently large. In the following, variables are renamed to obtain a *single assignment* program. Pre- and post increment operators, assignment operators, and function calls are recursively removed, hereby introducing new temporary variables.

The resulting *bit-vector equation* that consists of a constraint $\mathcal{C}$ and a property equation $\mathcal{P}$ are combined into a CNF (Conjunctive Normal Form) equation $\mathcal{C} \land \neg\mathcal{P}$ and checked for satisfiability by a SAT solver like Minisat [ES03]. If the CNF is unsatisfiable, the property holds.

2.3.5 The KeY Tool

KeY [BHS07] supports the formal development of object oriented software by integrating design, implementation, formal specification and formal verification. It enabled non-specialists in formal methods the use of formal artifacts and their understanding. Currently the verification of JAVA programs — or more precisely, a subset of JAVA called JAVACard — is fully supported. Specifications can be provided using either UML's Object Constraint Language (OCL) or the JAVA Modeling Language (JML). Specifications and proof obligations are fully automatically translated into a free variable sequent calculus for first-order dynamic logic which is checked using a deductive theorem prover.

The JAVA modeling language is not standardized by an organization and obtains most of its development through efforts by its community. The JML language is included in the JAVA sources in terms of a JAVAdoc comment, which may also serve a documentation purpose. By this approach it is relatively easy for developers acquainted with JAVA to write specifications. A detailed manual is found at [LPC$^+$08] and rich examples showing basic design decisions are treated in [LBR98].

The important keywords expressed by JML are denoted in the sequel. By the use of the `requires` keyword, the user defines the subsequent condition to be a part of the precondition of the contract. The final precondition is then computed as the conjunction of all preconditions that appear in the JML specification. In analogy, using `ensures` defines the following conditions to be included in the post condition of the contract. Since some of the elements of a JAVA class should be excluded from changes, the keyword `assignable` gives a list of elements which are allowed to change during the execution of the method.

The theoretical foundation underlying the KeY tool's verification engine is JAVACard DL [Bec01], an instance of dynamic logic (DL). What is special about JAVADL is that operators $\langle p \rangle$ and $[p]$ are used to build formulae, where p is a sequence of legal JAVACard statements. Formulas can be prefixed using the above operators. For example, $\langle p \rangle \phi$ denotes that program p terminates in a state in which ϕ holds. In contrary $[p]\phi$ does not require termination and stresses, if p terminates ϕ will hold in the final state.

2.4 Wireless Sensor Networks

The omnipresent phenomenon of our time is the increasing miniaturization. This trend is not restricted to special application domains. It is present in daily applications like mobile phones, laptops, et cetera. The performance of today's mobile devices exceed the capability of desktop-like computing hardware from a decade ago and this shift in paradigm is measurable for example through the transistors count of micro controllers. In the early 1960s the first main frame computers were equipped with a memory of up to 64 kB, the same that today's low energy miniaturized computers have.

And in fact, the measurable technological progress is not only limited to micro controller and their available memory. It also occurs from the perspective of networks and communication thus increasing the connectivity and offering completely new application domains. Essentially the network speed increased from some kilobytes using dial-up modems a decade ago up to high speed transmissions using broadband cable modems or the G3 standard for mobile devices.

It is obvious to state that this trend is fostered by mass marketing. So today's electronic devices grew from nice products to the market of disposable consumer electronics. This shift does not only allow every body to participate in technological change, it does also cause prices for electronic components to drop rapidly. And low prices in combination with the miniaturization and the connectivity open a new range of application domains, like wireless sensor networks: They are cheap in price allowing scenarios with hundreds of nodes. Furthermore nodes are capable of collecting environmental information including the execution of complex algorithms and since they are fully connected to other nodes they can share information with their neighborhood using the wireless link. In addition no infrastructure is required for communication since self-organizing algorithms are implemented that allow spontaneous action upon events or external triggers. All the above contributes to the view one should have on such a wireless network: a collection of autonomous systems that fulfill a common task by cooperation.

Today a vast number of useful applications settings exists for example like the surveillance of elderly people by health monitoring. Here, a dozen of sensors monitor the state of health through blood pressure, heart frequency, or other blood parameters. It is even thinkable that sensors in the shape of a pill can be swallowed to measure in-body parameters to complete the observable overall state of health. But this has still a visionary character and due to the low energy density of conventional batteries a challenge for future research.

What is already common is to use sensor nodes for environmental or structural monitoring of buildings, bridges, or other places that are hard to access. Also appealing in the context of wireless sensor nodes is the use for stock keeping purpose in companies. This is investigated for example in the use case of *smart drums* [KDD04]. In this scenario drums that contain chemicals are equipped with sensor nodes that allow in-situ monitoring of storage limits and indicate by an alarm if incompatible chemicals like reactive substances are brought too close together.

Up to this point wireless sensor networks seem to be a technological break-through and the technology of the future, but there are drawbacks involved. The power supply that these devices use is stored in a battery, a critical and limiting element. And in fact processor and memory grew exponentially in size according to Moore's Law, whereas the energy density of batteries did not do so. In contrary to the growth of transistors that double their transistor count about every 2 years, the energy density of batteries increases only linear. For example chemical Lithium-Ion batteries have an energy density of $0.3Wh/g$ (Watt hours per gram) and the energy storage of the next generation like Methanol driven fuel cells under development for several decades reach only up to $3.0Wh/g$. Since battery driven devices seem to enable unlimited visionary scenarios, the major restrictions are caused by the energy shortage that create a natural boundary on most of these dreams.

On the other hand the fact that nodes are using batteries brings them in the fortunate situation that they are not bound to a static place. Instead they can roam around and due to their wireless transceiver still forward collected data to the base station. Multiple applications fields exists and it seems that many visionary scenarios from years ago are now becoming feasible. As the most famous to name is the vision of small, smart, cheap and disposable devices that can be deployed as thousands of nodes like the prominent ubiquitous computing [Wei91, Wei88] setting. Here, these small mini computers are ominously present and very thoroughly integrated into everybody's life and activities.

2.4.1 Weakness of Wireless Sensor Networks

All of the above properties show the obvious strength but also the inferred weakness that is involved when using wireless sensor networks for real-life scenarios. In principle just the single matter that nodes employ a wireless medium for communication has many positive effects but in turn also causes additional constraints to consider.

In this sense transportation protocols must be robust to recover from channel errors. So appropriate mechanisms have to assure that a reliable transport of packets is guaranteed. This can be accomplished by the use of multi-path routing that can even withstand the complete loss of nodes on one particular route, but in turn requires an enormous overhead since every additional route in use increases the energy spending. In fact this strategy is seldom employed since it is conflicting to an energy preserving set up. More advisable is the use of a conservative approach that uses conventional packet checking algorithms. Hereby the energy use is evenly distributed. But for the normal use we rely on the network built-in resend mechanisms that detect corrupted packets and prevent data loss.

In contrary to wired network links, the wireless medium is affected by interference from external radiation sources. It could even be possible that an adverse entity is trying to stop the proper function of the network by a denial of service attack. In the wireless domain this is relatively easy to conduct by radio jamming i.e., the transmission of radio signals that disrupt communication by a decrease in the signal to noise ratio. If the network service has no built in counter-measure this technique can be used to drain the batteries and thus fully stop the operation of the network.

For a sensor node, the battery provides a limited source of energy. Although energy harvesting methods exists outside the classical wireless sensor network approach, that try to overcome resource constraints by generating power using ambient air temperature, the flow of air or vibrations, these techniques play only a minor role. Essentially these technological advances are worth to consider but they do only generate a limited amount of energy. For this reason the nodes mainly depend on energy preserving communications protocols and special algorithms developed for the wireless sensor node world that ensure a long lifetime.

Since wireless sensor networks are not only restricted to indoor use, they are often deployed outside where they can easily be manipulated, removed, destroyed or even reprogrammed by an adverse entity. Also tamper proof sensor nodes are available that hide their memory internals to unauthorized individuals, they are only deployed in small networks due to their costs. In this sense, the traditional view still considers physical access of an adversary on the devices. Furthermore it is important to consider intrusion attacks to define what an adverse entity can do and what not.

2.4.2 The Adversary Model

The model of an active *adverse entity* is adopted from the work of [Ben08]. It is able to intercept communication and is not limited to a small part of the wireless network. In detail, the adversary is able to capture nodes, bring them under its control by reprogramming and let them act according to its needs. Thus compromised nodes seem to behave legitimately and according to the protocol but all of a sudden they can act maliciously. From outside, adverse acting nodes cannot be distinguished from sound nodes since they behave protocol conform for most of the time unless they exhibit their fraudulent behavior which is visible.

Corrupted nodes collaborate to pursue the goals of the adversary [BCF07] during an attack and share their knowledge instantaneously with other adverse nodes using out-of-band mechanisms. In addition the communication between compromised devices is instantaneous in a sense that it requires no time. So the knowledge of a secret key to one compromised node is immediately known to all the nodes under the control of the adversary.

An important assumption is that only a fraction of nodes is affected by the adversary [BB08] and the majority of nodes are still sound. When neglecting this assumption and the adversary has for example more than half of the nodes under its control, it is able to manipulate all network traffic, and no trustworthy operation of the network is longer possible.

Essentially, the adversary's interest is to gain valuable information from the network that brings him some advantage. In consequence, it is only willing to take efforts up to a threshold beyond which it will no longer be profitable to him. In other words, if the attack is too costly, he will rather try to find other opportunities than starting an attack and infiltrating the sensor network. In this sense, taking over all nodes of the network would require much effort and time — maybe more than he is willing to spent — and surely narrows the profit of the network intrusion.

After some nodes in the network are compromised, the adversary has access to the wireless network. In addition we allow access to more powerful machines like a laptop like device or wide range antennas and hereby gain access to any region of the network. Due to the symmetric encryption the adversary can even interfere during the initial setup phase of the network when the aggregation tree is distributed and protocol parameters are passed around. But it is also limited, e.g., the adversary is not able to take control over the base station since this means that the user would not longer have access to sound data. In this way all information that the user would receive from the base station could be potentially forged.

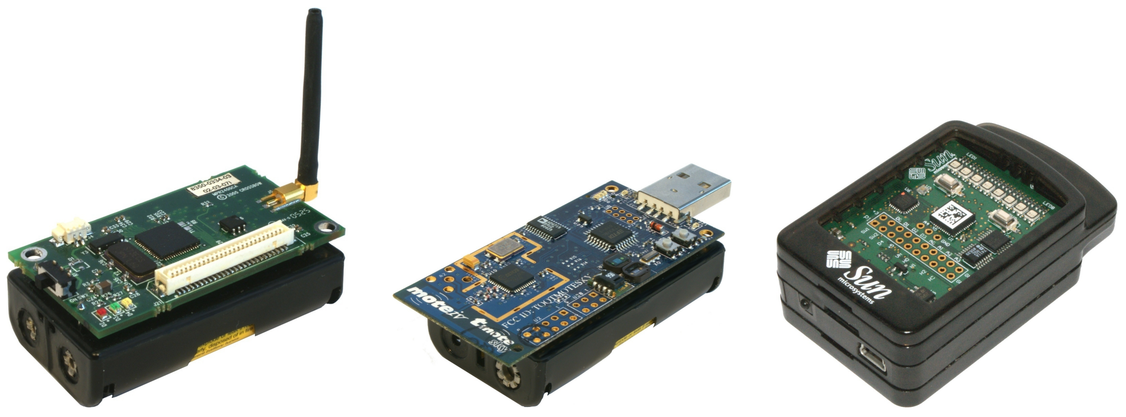
2.4.3 Sensor Node Platforms and Implementations

Today's sensor node devices typically consist of a micro controller with an attached memory that is separated into flash memory carrying the operating system and the executable program, and a volatile RAM to store runtime data. The source of energy is usually in the form of conventional AA-batteries. Many sensor nodes are able to collect environmental data like noise, temperature, light, humidity, etc. using their built-in sensor hardware. Through their attached transceiver they are able to communicate among each other, forming huge networks of thousand nodes.

TinyOS

TinyOS is an open source operating system for embedded devices and widely used in academia for the programming of embedded software. It has been developed at the University of California, Berkeley and receives today its main contribution through the open source community. The component based architecture of TinyOS and an event driven execution model make it very suitable for resource constrained hardware systems with respect to memory, computation power, energy shortness, etc. The energy efficiency goal is pursued by executing tasks from the task queue and then moving into an energy preserving mode where the highest portion of energy is saved.

The software developing platform TinyOS is offering means to program code for various hardware platforms. The node software is written in *nesC* a C dialect having special constructs for embedded devices. Furthermore it supports a variety of different platforms through a modular design. The *hardware abstraction layer* (*HAL*) consists of elementary components like transceiver or timer that represent the lower level of the hierarchical component model. In TinyOS a program is considered as a graph of components, of which each one has a frame and a structure of private variables. Three abstractions for components exists like commands, events, and tasks. Through commands and events



(a) MicaZ node by Crossbow (b) TMote Sky node by MoteIV (c) Sun SPOT by Sun Microsystems

Figure 2.3: Different node platforms

the inter-component communication is realized while tasks are used to express intra-component concurrency. What makes TinyOS very attractive is the hardware- and network simulator TOSSIM that is integrated into its environment.

TOSSIM [LL03, LLWC03] is a *discrete event simulator* for wireless sensor networks that is run on conventional PC hardware. Its primary goal is to simulate TinyOS applications with a high fidelity. Due to this, its focus is rather on simulation of TinyOS software and its execution than the simulation of realistic scenarios. In TOSSIM real world phenomena are abstracted like bit errors and no radio propagation model exists. Nevertheless, asymmetric links can be modeled by the use of directed bit error rates. By simply adding power draws for components, the simulator allows for thorough estimation of the energy consumption. Due to the solid integration into TinyOS the executable binaries can be directly derived from the nesC code for later deployment on real-life nodes or the software can be compiled, integrated, and emulated through the TOSSIM simulator.

Still, there exist some downsides that stem from the imperfect world mapping. On real nodes for example an interrupt can fire while other code is running where this is not possible in the emulator. In the emulation interrupts are treated non-preemptively due to the discrete event simulator, and this may cause the emulation to behave differently from real world scenarios. Last to name is the transceiver simulating the Mica networking stack with MAC, encoding, timing, and synchronous acknowledgements, rather than a simulation of the entire ChipCon CC100 stack, found in some of the nodes.

A vast selection of sensor node devices exists, developed by industrial companies and academic institutions. What they mostly have in common is their size, their restricted memory, and the low clock speed of the micro controller. But also differences exist in the attached sensor hardware, available memory and of course their market price.

MicaZ

The *MicaZ* (MPR2400CA) node (see Figure 2.3a) produced by Crossbow [CT, Cro05] is equipped with an 8-bit *ATmega128L* micro controller from Atmel [Atm] running at 8 MHz. Its CC2420 radio chip [Chi07] produced by TI (formerly ChipCon) is compliant to the IEEE 802.15.4 / ZigBee [Soc06] wireless standard allowing communication rates of 250 kbps. It even offers hardware based security mechanism through AES-128. The MicaZ has an internal programmable flash memory of 128 kB and a configuration EEPROM of 4 kB size. Its current draw is 8 mA while active and in sleep mode below 15 μA. It is programmable using its serial UART connection and in addition offers digital interfaces such as I^2C and SPI. Its 10-bit analog-digital converter (ADC) has a measurable input range between 0 and 3 V.

Sun SPOTS

Sun SPOTs [Sun08] (*Small Programmable Object Technology*) (see Figure 2.3c) developed by Sun Microsystems Laboratories are small mobile computers with a wireless network interface. Although often used, the terminology sensor network device is not appropriate and misleading since their main processor is a 32-bit ARM920T ARM RISC processor operating at 180MHz maximum internal clock speed. Two 16kB caches for data and instructions make it both fast, but at the same time very energy consuming. What is also exceptional is the fact that no embedded device OS is employed but rather a JAVA-Virtual Machine called *Squawk VM* that is shipped with a boot loader. The library itself is written mostly in JAVA and introduces special constructs for embedded applications.

Its memory consists of a 4MB flash and a 512kB pseudo-static random access memory. The ARM's operation modes are controlled using an additional power controller — an 8-bit Atmel ATmega88 micro controller. This micro controller wakes up the system from deep-sleep when an alarm occurs and vice versa, putting the ARM into sleep mode. In addition it also controls the bicolor LED, measures the battery voltage, the charge and discharge current.

On the communication board a CC2420 [Chi07] is used that is IEEE 802.15.4 compliant, and operates in the 2.4 GHz ISM unlicensed band with an effective bit rate of 250 kbps. Many peripheral interface units exists like USB host port, USB device port, Ethernet MAC, serial peripherals interface (SPI), universal synchronous/asynchronous serial interface (USART) and others. The sensors that reside on the board measure acceleration, temperature and light.

TMote Sky

The *TMote Sky* [Mot06] is designed at the University of California, Berkeley and distributed through MoteIV Corporation. They run an 8MHz TI MSP4300 micro controller having an on-chip RAM size of 10kB, an allocated program space of 48kB and a hardware protected external flash of 1MB. TMote Skys offer integrated humidity, light, and temperature sensors. In addition the nodes are capable of internal temperature and voltage monitoring. Their Chipcon Radio controller is IEEE 802.15.4 compliant with a mesh networking feature reaching transmission rates of 250kbps. The sensor nodes offer a number of integrated peripherals including a 12-bit ADC and DAC, and support I^2C, SPI and the UART bus

	minimum	normal	maximum
Supply voltage [V]	2.1		3.6
MCU on, Radio RX [mA]		21.8	23
MCU on, Radio TX [mA]		19.5	21
MCU on, Radio off [μA]		1800	2400
MCU idle, Radio off [μA]		54.5	1200
MCU standby [μA]		5.1	21.0

Table 2.3: Typical operation conditions of the TMote Sky node taken from the datasheet [Mot06].

protocols. The current draw of the TMote is taken from the data sheets and displayed in Table 2.3.

2.5 Summary

So far the necessary tools and the principles of wireless sensor networks are introduced. For the remainder of this work we investigate problems and questions that arise in the context of wireless sensor networks and strive for a solution by formal methods. Due to this reason, we now proceed with a new block, namely the application of formal methods to argue about the efficiency and energy consumption in wireless sensor networks.

In this sense, in the following chapter the energy use for sensor nodes is analyzed within a topology. To obtain energy related results, a UPPAAL model is defined in a first step and later on analyzed by the definition of appropriate specifications.

CHAPTER 3

Energy Efficient Routing and Scheduling

3.1 Introduction

The technique of model checking has been successfully used in many application areas. It has proved particularly useful in very early design stages when only a model or a blueprint of the product is available. Mainly contributing to its success is the exhaustive treatment of all states and possible interleavings. When involving and integrating formal methods within the development cycle of a system, it has proved that many flaws and errors can be revealed early, reducing costly changes in the later product design cycle.

Up to these days many case studies were carried out by using model checking techniques in this conventional way where one is interested in finding bugs and flaws in systems and model designs. The answer one is interested in has in most of these cases a dual character, namely *yes*, the system is safe. Or *no* if there is the possibility of livelocks, deadlocks, and errors which occur under special constraints revealed by the model checking algorithms.

A yet not very common application area evolved in the niche of constraint solving over the last years: the application of model checking techniques in the domain of reachability analysis. With this technique, the search mechanism of model checkers is used to find feasible and optimal schedules of a system instead of the discovery of hidden bugs.

The main advantage of this approach is that scheduling problems with changing parameter settings can be easily and clearly modeled and solved by means of reachability analysis. Especially *timed automata* which form a rich class of models has proven to be in particular useful for this kind of analysis, since the strategy of the shortest path is efficiently solved using the algorithms implemented in model checking tools like UPPAAL.

Moreover, once the automata model is defined, and proved to be a correct mapping to its real-world counter part, it is comparatively easy to propose queries or do constraint solving. By simply formulating the query in an appropriate specification language, the verification is automated and the model checking algorithm provides its results, which can either be a flaw of the system or a cost optimal schedule.

In the field of embedded real-time systems with limited memory and power constraints the idea of finding optimal schedules using the search algorithms of model checking is relatively new. When choosing the right degree of abstraction they can contribute to interesting new insights and surprising results.

In fact the analysis carried out by the use of model checking cannot be directly compared with modern state-of-the-art simulators like NS2 [Net], GloMoSim [ZBG98], etc. that analyze networks of thousand nodes including all intermediate layers with a nearly perfect abstraction. Formal methods will never replace simulation but its valuable exhaustive state search can round up investigations based on simulation. In this sense the outcomes of simulation and verification should be considered as complementary approaches. Tools based on formal methods can produce highly trust worthy data since these tools do have the indisputable advantage of being exhaustive and that their results are accurate and precise. Especially the distributed character of wireless sensor networks, where all interleavings need to be considered, suggests the application of the exhaustive search algorithms to this group of problems.

Related Work In the work of [FvHM07b] a medium access control protocol for wireless sensor networks called LMAC is modeled and verified. UPPAAL is used for the representation of the timed automata model, that has at most five nodes where all possible connected topologies are checked. The main property of interest is to detect and resolve collisions that occur when using several nodes for sending.

Model-based validation of quality of service parameters using biomedical sensor networks (BSN) is investigated in [TXY08]. The authors use a formal model based on timed automata to model temporal configuration parameters of the BSN to meet quality of service (QoS) requirements on network connectivity, packet delivery ratio, and end-to-end delay. The employed model uses timed automata that allow a study of dynamic reconfigurations of the network topology caused by physical movements. The verification results are compared with simulations in OMNeT++, a simulation tool for wireless sensor networks.

A model to compute optimal lacquer production schedules is introduced in the work of [BBHM05]. The authors show the feasibility to use techniques from the search mechanism of model checkers and apply them to reachability analysis. In this work the UPPAAL tool is used to find feasible and optimal schedules for storage costs, delay costs, etc.

In the work of [WS07, SW07] model checking is used to investigate a network of MicaZ sensor nodes using Timed Safety Automata. Using an energy computation cost-optimal schedules can be computed using the UPPAAL tool.

Overview This chapter is structured as follows: Section 3.2 will motivate the application of timed automata theory for the used setup. Section 3.3 explains the investigated wireless sensor network and the requirements that apply for the communication medium in order to gain a realistic scenario mapping. Thereafter Section 3.4 explains how the setting is mapped to the timed automata model and introduces the different network devices like *sensor nodes*, *routers*, and a *network controller* that the final composition will include. Furthermore means for the power draw are implemented that will enable the energy efficient cost estimation. The specifications tested against the model are denoted in Section 3.5 that represent safety and liveness properties, each model has to fulfill a desired degree of realism. Results of the analysis are discussed thereafter in Section 3.6 with the focus on a reference sensor end device, and a router. The conclusion then summarizes the work, reflecting the formal approach with timed automata and giving the experience learned from the experiments with UPPAAL.

3.2 Modeling Method

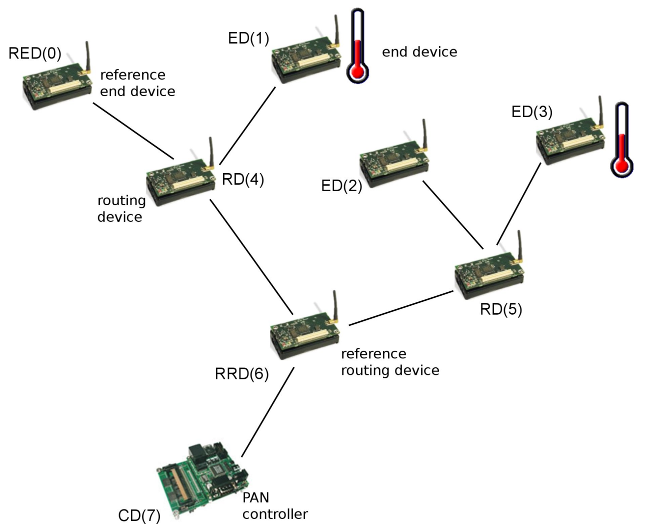
Having a concrete idea of the problem instance in mind, the question that needs to be answered is which tool is best suited for a networking scenario where nodes of different kinds interact to fulfill a common task. One of the aspects that is surely targeted is the use of energy over different sleep modes which is strongly related to time. Therefore the model should have a notion of time and a good candidate for this task is the UPPAAL tool (see Section 2.3.2) with its underlying timed automata concept. Although also probabilistic aspects like transmission errors — that occur with high probability noisy areas — would be an interesting point of research, the drawback of probabilistic tools overweight here: no traces to the satisfying states are provided and they lack constructs to denote time which is essential when considering a system with limited resources. Due to this reason we decided to use UPPAAL.

UPPAAL is a fairly efficient and integrated tool environment for the design, simulation, and verification of real-time systems. Although the simulation cannot be compared with modern network simulators it has still proven very useful during the design phase since it gives the user the possibility to validate the model and check it for plausibility. The tool is eminently adequate for systems that can be modeled as a collection on non-deterministic processes. For communication and inter-process message passing either shared variables, binary-, and broadcasting channels can be used, combining a finite control structure with the use of real-valued clocks. Furthermore, it is easy to use since the verification algorithm works fully automatically, and in case a state satisfying property φ is found, a trace reaching that very state is provided. By different strategies like "shortest" which returns the shortest path in terms of automata-steps, or "fastest" returning the shortest path with respect to time, the user can select the best suited solving-strategy.

3.3 Sensor Network Scenario

The analyzed model represents a wireless networking scenario in which three different kinds of sensor nodes perform a common task. Considering the illustrations in Figure 3.1a, one possible routing tree at a transmission strength of $-10dBm$ is indicated by the lines interconnecting the nodes. The task could be for example to collect a temperature reading at the end devices and forward it using the routers to the network controller positioned at the sink where the user can access the readings. This scenario will then be evaluated with respect to different properties that will be described in more detail in the specification Section 3.5.

From the different topologies that exist, the mesh network seems to be most appropriate for the selected scenario, since it is versatile and offers a wide range of scenarios in contrary to the other settings (e.g., star topology, cluster tree, hybrid architecture). In the considered case each node can act as a dynamic router and communicate directly with other nodes without the use of a central entity. We consider the network in non-beacon mode where periods of sleeping and awakening are asynchronously spread over time. This has the



(a) A sample scenario of a sensor network with reference end device (RED), end devices (ED), reference routing device (RRD), routing device (RD), and the network controller device (CD) at a sending strength of $-10dBm$.

$$\begin{pmatrix} 0&4&8&8&2&6&4&6\\ 4&0&8&8&2&6&4&6\\ 8&8&0&4&6&2&4&6\\ 8&8&4&0&6&2&4&6\\ 2&2&6&6&0&4&2&4\\ 6&6&2&2&4&0&2&4\\ 4&4&4&4&2&2&0&2\\ 6&6&6&6&4&4&2&0 \end{pmatrix}$$

(b) Distance matrix for the selected scenario carrying unit-less distances entries.

Figure 3.1: The model description with the scenario and the model's internal representation of the distance matrix.

big advantage that the interesting observations about collisions and individual sleeping periods can be investigated. In contrary the beacon mode is fully coordinated and a super frame is broadcasted by the PAN coordinator on which nodes can synchronize. In the considered scenario each node decides autonomously upon its current state which action to pursue next. Consequently devices are able to wake up independently of each other in certain intervals, collect information from their surrounding, communicate with proximity nodes and fall back to sleep.

Since most energy is preserved in sleep mode where processor and on-board transmission unit are shut down, we target an average duty cycle of 1% [LO05] by setting model variables `ActivePeriod` as 1 and `ActiveCycle` as 100. Explicitly note at this point, that the model is beacon-disabled without a contention free period i.e., collisions can always occur. Distances between respective entities are modeled using the distance matrix as shown in Figure 3.1b to determine the flow of packets within the network. Further, the number of hops until a packet arrives at its destinations is determined by the entries of a distance matrix. Values herein can be changed but stay fixed for the selected experiments.

3.3.1 Modeling the Communication Medium

The communication medium is modeled as one channel over which all devices transmit and receive. Hence each device has, according to its selected transmission range, a certain range in which it can receive packets.

The ZigBee protocol [Zig05] or the IEEE 802.15.4 [Soc06] stack are real world realizations which are considered as role guideline for the here developed model. The inter-device communication is realized using the CSMA/CA (*Carrier Sense Multiple Access with Col-*

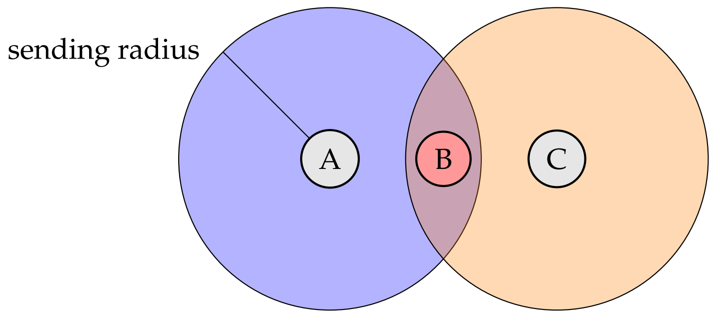


Figure 3.2: The hidden terminal problem.

lision Avoidance) access mechanism over which all devices send and receive. That is in principle if a device has data to send, it will listen on the channel for ongoing communication. In case the medium is found to be idle, the node will send data and wait until an acknowledgement from the recipient are received.

As the transmission power is increased for each device more nodes can be reached at the expense of energy. And although packets can be delivered faster since they need less hops, the number of collisions especially in busy networks increases. In addition, the effect of the so called *hidden-terminal problem* will increase and contribute to packet collisions. This issue also known as *hidden-station problem* refers to a setting in which hidden nodes simultaneously try to communicate with a shared and visible node. Since the two hidden nodes cannot detect each other (node A and C in Figure 3.2), meaning they cannot sense ongoing transmissions on the channel, they will concurrently sent, causing a collision at the receiver's side (node B).

An increase in the sending strength might be advantageous under some constellations, although it might also harm in others. Although the model makes no use of changing transmission rates during execution due to complexity considerations, the ranges can be changed for each execution in discrete steps. Possible rates are $-10dBm$ for the low transmission range sending over 2 distance units, $-5dBm$ for a range of 4, and the maximum sending strength at $0dBm$ with a range of 6 units. The unit-less sending range is related to the distance matrix which contains the distances between nodes and hereby denotes the scenario. The investigated topology is stored in the distance matrix of Figure 3.1b. To account for a very restricted state space, the model's transmission rates are constant but changed for different properties to imitate different scenarios. This means that the sending strength is fixed by setting it to either low, medium or high during the analysis of a scenario. Note that the reception rate has no such parameter and is hence kept constant.

3.4 A Timed Automata Model

In order to obtain a deeper understanding of the timed automata model, functions and essential variables are explained. Each automaton is labeled with a unique ID to make instantiations of the same template distinguishable. In addition there is a global clock `t` which can be accessed by each process, giving the model a notion of time.

In the automaton shown in Figure 3.3b nodes labeled by `Down`, `Send`, `Idle`, and `Rcv` are examples of locations. The initial location — this is the leftmost — is recognizable by the double circle. Whenever the globally modeled clock exceeds a value where a new round should be started (`t>=ActiveCycle`), the controller is initiating a new cycle by waking up all devices from sleep-state through broadcast of action `GoIdle!`. The out-going edge in Figure 3.3c from the initial node carries the guard `t>=ActiveCycle` which means that the transition is enabled when the guard is true. It does not say that the edge must fire as soon as t gets greater than ActiveCycle. By the additional invariant `t<=ActiveCycle` on the `Down` state, the automata is allowed to stay in the down state as long as the invariant is not violated. The behavior resulting from this combination of guard and invariant forces the automaton to leave the state `Down` exactly at time `ActiveCycle`.

As a consequence, all devices wake up and all reference devices update their energy consumption by calling the cost energy function `CE()`. In addition, the end devices (see Figure 3.3a) collect sensor values which are queued (`q=(q+1)%MOD`). From now on, each device is allowed to process and transmit data individually until the active period expires (`t<=ActivePeriod`). Note at this point that the router and controller have an active period prolonged by one time unit to account for network management and configuration messages. All devices have to leave state `Idle` whenever the state invariant (blue and bold labels in Figure 3.3) labeling the idle state is violated. Sensor nodes can fall back to sleep-mode earlier before `ActivePeriod` expired, that is if their sensor value is successfully transmitted to the network.

Much of the modeling is encoded in functions to retain a human-readable system design. The main functions that were used are function `CheckID(id)` which checks whether any device of `id` is within the receivers range in which case `true` is returned. Functions `Clean(id)` and `CleanAll(id)` take care for a reset of the channel to the idle state denoted by `-1`. The difference is that the first function is used after a sending attempt during which the channel was sensed busy for clearing all channels with identity `id`, where `CleanAll` clears all channel entries. Finally, by the use of `CheckAv` the availability of other nodes is inspected. This complicated modeling of the channel is required since in reality a signal once sent by a node is fading over time. In contrary, the channel in the model requires an explicit reset to an idle state or otherwise it will stay busy forever. The sources of the function bodies can be found in Appendix A.1.

Communication Medium The medium is modeled in UPPAAL as follows: The array `a[]` of length N denotes the availability of sensors and is manipulated by the transition updates during the operation of the network. For each i, $0 \leq i \leq N$ the array entry `a[]` will be one of the values in $\{-1, 0, \ldots, N-1, N\}$. The intended meaning behind this is, that `a[i]==j` if and only if a sending sensor j intends to send, and a node i is within the reception range of j. In addition variable `a[i]==-1` is set to signal that the channel within the proximity of sensor i is idle. Variable `a[i]==N` is assigned to indicate that a collision occurred at node i caused by any of the neighboring sensor nodes on the channel.

3.4.1 Sensor Network Devices

In the selected approach of building a sensor device model using timed automata, homogeneous nodes are used which are all equal in their capabilities. Also the nodes are in charge of different tasks they have to fulfill. They can, for example, be implemented on the same hardware, like the well known MicaZ platform [Cro05].

Three different node tasks will be modeled, that is the end device *ED* that will collect environmental information and transmit it to the network. The routing devices *R* that interlink different nodes and route information through the network until the sink is reached. The base station or network controller *CD* will manage the network activities and all actions that take place, i.e., decide upon the sleep phases of the sensor network. The corresponding UPPAAL models are depicted in Figure 3.3. Obviously, there are only subliminal differences between the three templates.

To reduce the complexity of the model and thus increase the number of possible instances that can be checked, nodes are modeled to contain only their characteristic parts which are needed to fulfill their task. This means, that the end device does not need a receiving part since its main task is to sense information and broadcast it to the network.

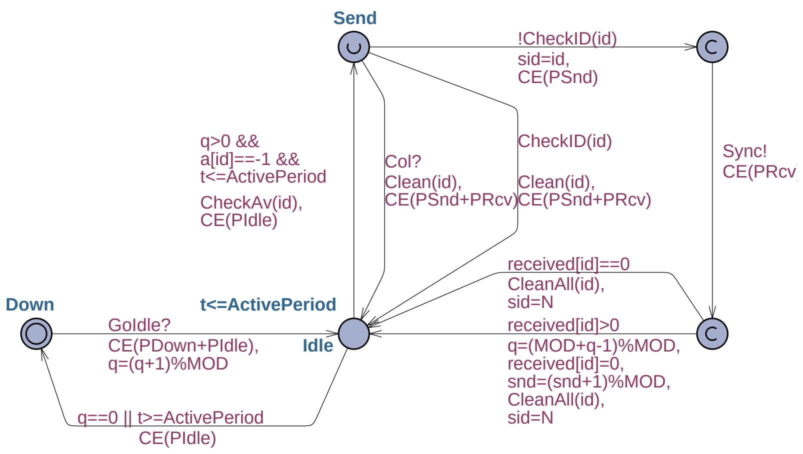
End Device

The sensor nodes in Figure 3.3a collecting data from the environment are the only devices that have the capability to collect sensor values, but in turn, they have no mean to receive packets from the rest of the network. From state `Down` they become active with the `GoIdle` signal issued by the network controller and move to state `Idle` where they stay either until their timer expires (increasing above `ActivePeriod+1`) or until their queue is emptied (`q==0`). A queue becomes empty by sending its content to the network and it is filled with a packet when waking up and moving to state `Idle`.

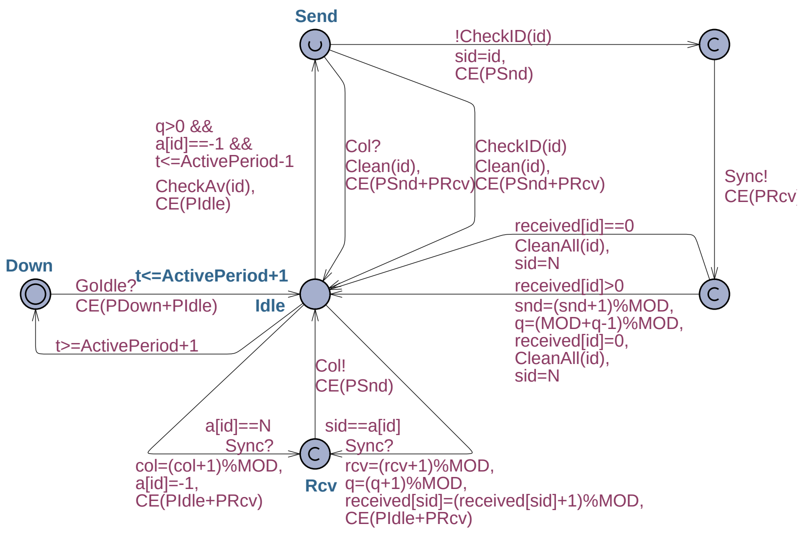
The activity of sending occurs in several stages. If a node has a packet to transmit, it is taking the transition to state `Send` if the following guard constraints are fulfilled. There is a packet in the queue `q>0`, there is no ongoing communication nearby indicated by variable `a[id]==-1`, and the time since the last sleep phase is still less or equal to `ActivePeriod-1`. When the transition is taken, function `CheckAv(id)` is checking whether any nodes in the neighborhood are available for the reception of packets. After the channel is sensed, the transition to state `Send` is taken and by this the automata checks for ready to receive nodes in the scenario. Ongoing transmissions are sensed by function `CheckID(id)`. This is done by checking whether any device within range of device ID — expressed by `dist[id][i]` — exists in which case `true` is returned. If the node has no exclusive access on the channel and no nodes can be reached by the preset range, `false` is returned and the automata returns back to state `Idle`. Otherwise the node sends its data to nodes within range in state `Idle` using action `Sync!` and waits for acknowledgements. In case the packet is acknowledged it is removed from the queue, the `snd` counter is incremented, the queue size `q` is decremented, and the channel is cleared for the node with ID using function `CleanAll(id)`. In the other case that no acknowledgement is received, the node will also mark the channel idle again by calling `CleanAll(id)` but its packet will still stay in the queue. Essentially variables `snd`, and `q` will need no update, since the packet could not be sent. This describes the behavior of the sensor node.

Network Router

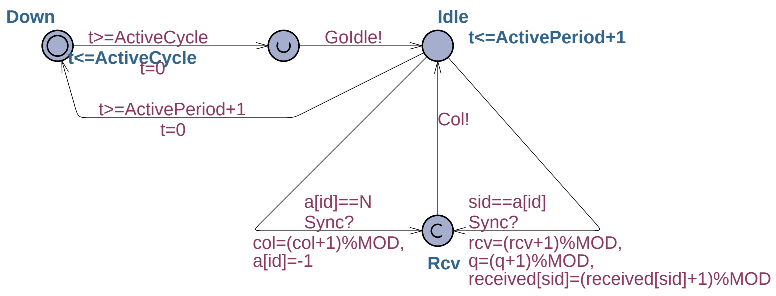
The network router depicted in Figure 3.3b is interlinking nodes and is hence required to send and receive packets. Since the sending part is similar to the sensor nodes, the receiving mechanism will be described in the following.



(a) Template for reference sensor node



(b) Template for reference routers



(c) Template for network controller

Figure 3.3: UPPAAL timed automata model of the sensor network.

Devices are only able for the reception of packets while being in state `Idle`. In this state they synchronize on the signal `Sync!` broadcast by a sender and move to the receiving state `Rcv`. Notice that there are two transitions possible, that is the left transition which represents the occurrence of a collision denoted by the array `a[id]`. Since this value equals to `N` there are more than one sending events and consequently the node increments its collision counter (`col=(col+1)%MOD`), clears its array entry and computes the cost for this failed reception attempt.

The right transition represents a successful reception of a packet. This is because no sender is interfering with the ongoing transmission and thus `sid==a[id]`. Upon this, the reception counter is incremented (`rcv=(rcv+1)%MOD`), the packet is enqueued (`q=(q+1)%MOD`) and the acknowledgment is sent back to the sending process. Modeling the acknowledgements is done through array `received[sid]` where the respective position is incremented (`(received[sid]+1)%MOD`) for each reception of a sensor node the value in the array is incremented.

Network Controller

The remaining template belongs to the network controller displayed in Figure 3.3c which is always uniquely present in the scenario. The controller is placed at the sink through which a user can access the network. Due to complexity, this device is only equipped with a receiving part which is similar to the receiving side at the router. So it will not be described in detail.

3.4.2 Energy Cost Estimation

Since one of the goals is to find optimal cost schedules with the modeled automata, realistic energy values need to be incorporated into the model. For being comparable to results obtained from simulation and real time evaluation in a testbed, the energy draw of the MicaZ sensor nodes (see Table 3.1) is used due to several reasons. First of all, these devices — which are manufactured by Crossbow Inc. — provide a versatile platform for low-energy sensor networks because they can beneficially be used in combining low transmission rates while behaving energy conscious. Especially in academia where Crossbow is showing presence on conferences and workshops the popularity of the Mica-Mode family gained popularity and can be seen as the reference platform.

The power draw of the MicaZ nodes [Cro05] is incorporated into the model, using the values shown in Table 3.1. Every relevant state change within the model is accompanied by a call to the energy function `CE()` (see Figure 3.4) which accumulates the energy use.

The radio strength is given in dBm, a unit that relates the sending power level to an absolute magnitude of $1mW$, a common used dimension in the field of high frequency radio techniques [Win05]. In consequence when relating the performance ratio of the sending power P_1 to $P_2 = 1mW$, the resulting sending strength can be computed:

$$P = 10 \cdot lg \left(\frac{P_1}{P_2} \right) dBm = 10 \cdot lg \left(\frac{P_1}{1mW} \right) dBm \tag{3.1}$$

As mentioned before function `CE()` accumulates costs from Table 3.1 to account for state changes of the automata. Respectively adopted to this design is the cost of leaving

```
void CE(int e){
     //Compute Energy of Mote modulo CMOD
     c = (c+e) % CMOD;
}
```

Figure 3.4: Definition of the UPPAAL cost function.

state `Down`. Using cost as a product of `PDown` and 99 time units gives an accurate energy consumption for the time spent in this state, totaling the transition costs to `PDown' =` 1 547μA. This approach is used since real valued clock values cannot be incorporated in arithmetic calculations in the present UPPAAL version 4.0.3 at hand from October 2006.

To avoid an unnecessary blow-up of the state space, not every device model has a cost estimation function *`CE()`*. That means in particular that only one single reference sensor node (see Figure 3.3a) and one reference router presented at Figure 3.3b are used in the system of distributed timed automata. The decision not to include a cost estimation function into the controller node is motivated since this sink has an unlimited power supply.

For the retrieval of proper estimates to account for the energy used, it is required that UPPAAL uses the cost from Table 3.1 multiplied with the corresponding time spent in each of the states. Unfortunately due to type constraints of the involved figures, this did not lead to the expected results in the present program version. In particular, when considering the multiplication of energy values with the corresponding time, a type mismatch occurred. The workaround that was used is to precompute the energy needed for sending, and receiving of a packet depending on the actual transmission strength, a proper estimate is found. Also the energy for the idle state is estimated by the product of the idle energy draw and the time spent in the idle state over approximated by `ActiveCycle`. By integrating this power draws in the model, an energy computation is albeit the variable type constraint still feasible.

Besides this, the use of *priced timed automata* [BLR05, RLS04b] as proposed in many studies [BBHM05] is also investigated. The application of UPPAAL Cora which is using this technique of cost optimal reachability analysis in linearly priced timed automata did not deliver the desired results and completely failed in the example at hand. Although several case studies did provide promising results and hinted the application of this tool, it failed to do so for the selected application scenario. Albeit its utilization in simpler cases, it failed to do when working with increased complexity as in the selected example.

3.4.3 State Space Reduction

Several efficiency means were introduced to reduce the complexity to a degree which finally enabled the verification of queries. For example many of the variables are declared in combination with the UPPAAL keyword *meta*. Consequently these variables will not be part of the state vector spanning the state space. Note that this construct needs careful handling for the following reason. If a variable is essential for the model to properly function but not included in the state vector, two snapshots of the automata in which the model is in different states could be mapped to the same state in the state space. Although

state	processor $[\mu A]$	transceiver $[\mu A]$	remarks
PDown	15	1	energy draw in sleep
PSleep	8 000	1	MCU up, TX/Rx down
PIdle	8 000	20	MCU up, TX/Rx up
PSnd1	8 000	11 000	sending at $-10dBm$
PSnd2	8 000	14 000	sending at $-5dBm$
PSnd3	8 000	17 400	sending at $0dBm$
PRcv	8 000	19 700	receiving mode

Table 3.1: Energy consumed by the MicaZ Sensor in each state.

the simulator correctly represents the model behavior, the verification will not do so and properties about the state will be falsely interpreted.

The reason for using constant MOD stems from the requirement imposed by all model checking approaches that the number of states should be finite. Hereby it is possible to limit natural numbers m involved in all arithmetic operations by the use of the modulo operation $m\%MOD$. As a result, the numbers are within the range of $0, \ldots MOD - 1$. Any kind of counting — in the present application that is the number of collisions, number of packets sent or received and also the energy consumption — has to be truncated in that way.

Another way to reduce the state space is by omission of variables not required by the present analysis. These can for example be commented out. Similar to this, the reference models of the sensor node and the router are only included in the model if they are important for the actual property. In case that one is not interested in energy and just concerned with other properties, the energy related variables and means can safely be neglected.

3.5 Specification

The properties will be formulated in the querying language CTL (see [Kat03]) which is suited for the analysis of real-time systems. Although UPPAAL will only allow the use of a subset of the expressiveness of CTL — that is a combination of path and state quantifiers without nesting of temporal operators — still many useful and interesting properties can be formulated in the available querying language.

Before the model is analyzed using the energy-related measures, it is necessary to assure the model's correctness by verifying certain properties. For sanity checking it is required that the following state formula holds. It describes the safety criterion as the absence of deadlocks in terms of the CTL formula

$$\mathbf{A}\Box\, no\, deadlock \tag{3.2}$$

The keyword `deadlock` is a built in construct in UPPAAL, describing a state of the model in which no further transitions are possible. If deadlocks would exist, the model could get stuck and become inoperable for the analysis.

3.5.1 Reachability

In addition, the following *reachability* properties have to be true, that is that certain states are reachable. In the selected scenario possible properties of this form are for example "Does there exist a state where the controller CD receives a packet?" The formal equivalent query is

$$\mathbf{E}\Diamond\ CD(7).rcv = 1 \tag{3.3}$$

In addition each end device in the network must be able to send a packet. For the reference end device RED this can be states as:

$$\mathbf{E}\Diamond\ RED(0).snd = 1 \tag{3.4}$$

Since reachability testing has the incentive to perform sanity checks on the model, more properties with respect to the actual model behavior need to be defined. Especially the properties often cited as safety and liveness properties are of special interest in this context.

3.5.2 Safety

In principle, a *safety property* stresses that nothing bad will ever happen. This could for example be that it is not possible for a node to block the channel for ever and thus prevent others from accessing it. We formulate such a safety property in a positive way, namely that something good is invariantly true.

One property that should always hold involves the transmission range of nodes. To test it the following setting is considered: Using a transmission range of 2 spaces the controller $CD(7)$ can only receive packets from the router $RD(6)$ since all other nodes are not within its range (see Figure 3.1a). This means that no packet collisions can occur. When switching now to a higher transmission range collisions will emerge due to the hidden-terminal effect. To test for this situation the following property is specified and verified with different transmission ranges.

$$\mathbf{A}\Box\ CD(7).col = 0 \tag{3.5}$$

3.5.3 Liveness

Essentially for a sound model are the *liveness* properties, stressing that eventually something good will happen. Since UPPAAL forbids nesting of operators, is has a special construct to account for response properties. To express properties like *whenever ϕ is satisfied then eventually ψ will be satisfied* the UPPAAL construct $\leadsto$ (written as -->) is added to the UPPAAL query language to denote *leads to* or *response* properties.

The property whether there is message passing on the channel is fundamental for the later analysis. To check that the channel is utilized and does not stay idle is expressed in the querying language as:

$$\mathbf{A}\Diamond\ sid = -1 \tag{3.6}$$

Also important for the proper function of the model is the property that a node will not block the channel forever which is similar to the above property. As such the situation that sensor node with ID 3 will not block the channel forever can be formulated in CTL as:

$$\mathbf{A}\Diamond\ sid = 3 \tag{3.7}$$

Another important issue is the reliable packet transport. That is exemplified in terms of the UPPAAL model that if a packet is sent, it will eventually be received by another node. In particular, it should hold that if node $RED(0)$ has sent a packet that collides with packets sent by other end devices (note that only end devices can initiate packets in the selected scenario), this packet will eventually arrive at the sink.

$$\begin{aligned}(RED(0).snd = 1 \wedge ED(1).snd = 0 \wedge ED(2).snd = 0 \\ \wedge ED(3).snd = 0) \leadsto CD(7).rcv = 1\end{aligned} \tag{3.8}$$

The following property states that if a packet is queued, it is either sent, or a collision occurs

$$RED(0).q = 1 \leadsto (RED(0).snd = 0 \vee R(4).col = 1) \tag{3.9}$$

Also considered bad is the property that messages will get lost in the network, and in consequence the reliable transfer of messages must be guaranteed. This is formulated as

$$\begin{aligned}&(R(4).rcv = 0 \wedge RED(0).q = 1) \leadsto \\ &((RED(0).snd = 1 \wedge RED(0).q = 0 \wedge R(4).rcv = 1) \\ &\vee (RED(0).snd = 0 \wedge RED(0).q = 1 \wedge R(4).rcv = 0))\end{aligned} \tag{3.10}$$

Additionally to the above specified properties the model is used for the computation of energy requirements. Since the controller is assumed to have unlimited energy resources it will not be considered for the analysis of the power use.

Sensor nodes are analyzed in the following experiment by the use of a reference sensor node $RED(0)$. The following property is of special interest: "How much energy does the reference node spend by sending a packet ($RED(0).snd = 1$) routed through the network and eventually received by the controller ($CD(7).rcv = 1$)?" This property is expressed in the CTL query language of UPPAAL as

$$\mathbf{E}\Diamond\ CD(7).rcv = 1 \wedge RED(0).snd = 1 \tag{3.11}$$

By the use of the temporal operator $\mathbf{E}\Diamond\varphi$ ("Does there exist a path such that φ eventually holds?") the path is returned which satisfies this property and since the strategy which returns the shortest path is chosen, this will even be cost optimal.

Routers After having studied the energy use by sensor devices a further step is to investigate the costs that occur at the routing devices since they need more power due to higher activity. For this scenario, the reference router $RRD(6)$ from Figure 3.1a has been chosen, since it interlinks the controller with the rest of the network, and is hence most critical to energy constraints. The analysis observes the energy consumption by the router using different transmission rates and numbers of collisions. These properties are stated in the following specifications 3.12 to 3.15.

$$\mathbf{E}\Diamond\ CD(7).rcv = 1 \tag{3.12}$$

$$\mathbf{E}\Diamond\ CD(7).rcv = 1 \wedge RRD(6).col = 1 \tag{3.13}$$

$$\mathbf{E}\Diamond\ CD(7).rcv = 1 \wedge RRD(6).col = 2 \tag{3.14}$$

$$\mathbf{E}\Diamond\ CD(7).rcv = 1 \wedge RRD(6).col = 3 \tag{3.15}$$

3.6 Results

In the following the results from the verification are discussed in detail. Properties are verified using a hash table size of 512MB for state hashing in UPPAAL. The format of the diagnostic trace which is important for the energy related tasks is selected as shortest path, since the major goal is to find cost optimal energy consumption for queries.

Before the outcomes of the energy requirements are computed, the model is checked for possible errors introduced into the model by validating the CTL queries 3.2 to 3.5 as noted in Section 3.5. Fortunately all of the above statements are verifiable, and the results are as expected. This holds also for Property 3.7 which is invalid. Otherwise this would mean that the channel can be blocked forever by a node, and no further communication would be possible. This would clearly indicate a modeling error.

For Property 3.5 the situation is changing with increased transmission range. Where at a range of 2 and 4 the property can be confirmed, it is invalidated with a transmission range of 6 which can be explained in a natural way as follows. At a range of 2 all communication to the sink directly flows over the router $RRD(6)$. When increasing the range to 4 there are two more devices — that is router $RD(4)$ and $RD(5)$ — which can talk to the controller. Since $RD(4)$ and $RD(5)$ are also separated by 4 spaces, they can hear each other and due to the CSMA scheme they will not send colliding packets.

This situation now turns when increasing the range to 6 units. In this situation even $RED(0)$ and $ED(3)$ can directly send packets to the sink since they are only 6 spaces away. But their distance among each other is 8 spaces and in consequence they cannot hear each other when sensing the channel for ongoing transmissions. Up to this point the model at hand fulfills the demanded networking characteristics and suggest that the timed automata model is sound. This is also proved using formal means of model checking. The results are summarized in Table 3.2 for different transmission ranges.

Thereafter, the model is used for a deeper analysis since the model at hand seems to fulfill the demanded networking characteristics, i.e., it is free from deadlocks. In addition

	sending strength		
property	2	4	6
3.2	valid		
3.3	valid		
3.4	valid		
3.5	valid	valid	invalid
3.6	valid		
3.7	invalid		
3.8	valid		
3.9	valid		
3.10	valid		valid

Table 3.2: Liveness, safety and reachability properties verified with UPPAAL.

distance	TX[dBm]	power use [mA]
2	-10	64.3
4	-5	67.3
6	0	70.7

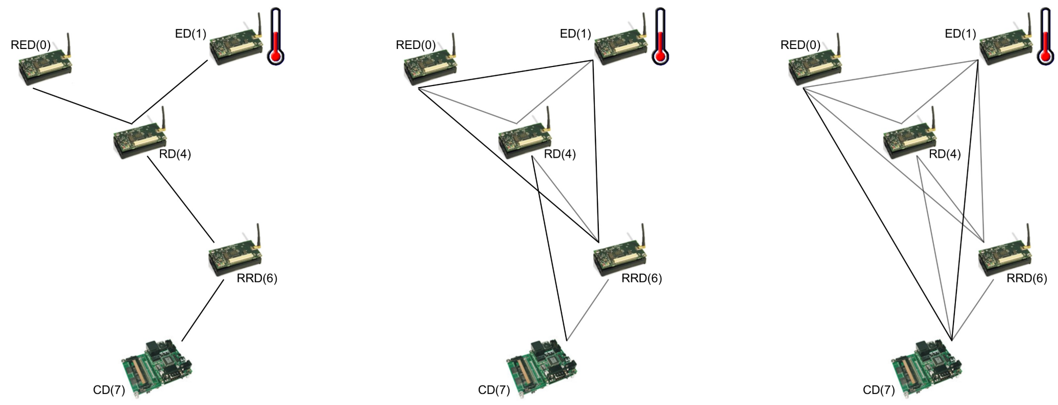
Table 3.3: Energy consumed by the sensor device for different transmission ranges for property $\mathbf{E}\Diamond\ CD(7).rcv = 1 \wedge RED(0).snd = 1$.

we showed, that the controller is able to receive a packet and sensor nodes can send packets. As in real networks it is also possible that collisions can occur, but a reliable data transfer is guaranteed meaning that no packets are lost. In the sequel, the energy estimation is used with the model described above.

Sensor End Devices and their need of energy is investigated in the first analysis. The results from specification 3.11 are as follows. As expected the energy is increasing from $64.3mA$ at the lowest sending strength over $67.4mA$ up to $70.7mA$ for the highest transmission setting (see Table 3.3). Hence for a packet to send, the sensor node requires more energy the higher the selected transmission strength is. The variation in these results can be explained by referring to the sending energy as shown in Table 3.1 which was incorporated into the model. So the discrepancy in the energy use stem exclusively from the different transmission draws in the sending state. In particular, the energy draw in the idle state, in the sleep state and for reception stay the same.

Routers do require more energy than the end devices since they have more packets to send. To investigate this, Property 3.12 to 3.15 are analyzed — that is how much energy has router $RRD(6)$ to spend in for a cost optimal routing — with the following outcome (see Table 3.4). For readability Figure 3.6 depicts all properties with their resulting energy draw.

As anticipated, the analysis for Property 3.12 — where no collisions occur — shows that a cost optimal path which satisfies formula $\mathbf{E}\Diamond CD(7).rcv = 1$ exists. Figure 3.5 depicts two scenarios with different transmission settings and their corresponding routing trees.



(a) TX strength of $-10dBm$ (b) TX strength of $-5dBm$ (c) TX strength of $0dBm$

Figure 3.5: Network showing routes for different transmissions. The gray lines indicate links already present when using a smaller sending strength, where the black lines show links only possible with the current strength.

Depending on the transmission setting this figure is increasing when switching from $-10dBm$ to $-5dBm$ but then rapidly falls at $0dBm$ to $46mA$. The explanation for this is at the lower sending level the energy expenses at the router increase, since the sending state now requires more energy. Instead, if the transmission power is set to maximum, the router does no longer need to forward packets, since the router $RD(4)$ can now directly communicate with the network controller device at the sink. Hence the required energy for the router $RRD(6)$ reduces if all devices send at a strength of $-5dBm$.

When considering the second query (formula 3.13) with one emerging collision the situation is changing. Due to the collision occurring at the routing device $RRD(6)$ more energy is needed. But with increasing sending strength the energy use drops for a similar reason as above, namely that more devices can directly talk to the network controller. Notice that this scenario only considers the cost optimal routes for the reference routing device. For other nodes in the network this means that their expenses on energy increase since they have to spend more energy on their communication to reach directly the sink and hence this figures have to be treated with care.

Further increasing the number of collisions up to three keeping the transmission strength constant to $-10dBm$, the energy draw is also increasing. This does not need any further explanation. Just by an increase of the transmission range and hereby the number of nodes that are reached and keeping the range constant to four, it is peculiar that a minimal energy level exists which is even the same for 1 and 2 collisions.

In general it is important to say that when increasing the number of collisions this does not necessarily imply that hereby also the energy use will increase as shown in the middle column of Table 3.4. For a more detailed and illustrative view refer to the graphical representation at Figure 3.6.

distance	2	4	6
TX $[dBm]$	-10	-5	0
property	power use $[mA]$		
$\mathbf{E}\Diamond CD(7).rcv = 1$	120	126	46
$\mathbf{E}\Diamond CD(7).rcv = 1 \wedge RRD(6).col = 1$	156	104	107
$\mathbf{E}\Diamond CD(7).rcv = 1 \wedge RRD(6).col = 2$	211	104	168
$\mathbf{E}\Diamond CD(7).rcv = 1 \wedge RRD(6).col = 3$	266	162	229

Table 3.4: Energy consumed by the reference ZigBee router under different scenarios.

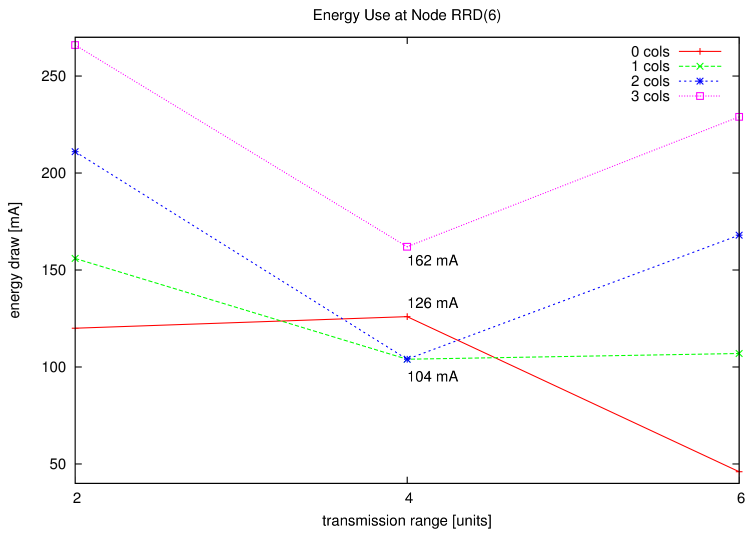


Figure 3.6: Results from the energy measurement using the timed automata model.

3.7 Conclusion

The use of formal methods and the application on timed automata models as used in this chapter shows the feasibility of this technology in the world of wireless sensor nodes. Especially the sound basis that was developed in terms of the UPPAAL model is a good start to reflect arbitrary scenarios. That is in particular the use of the distance matrix for modeling topologies, the use of different types of nodes by embracing the energy model, and the sensing mechanism for the channel. The use of the querying language of UPPAAL allows various situations to be analyzed, combining the exhaustive search algorithms of model checking with the visual and human readable representation in terms of the timed automata model.

The conducted analysis shows that the timed automata model presented here are a good start for the analysis of energy consumption of sensor devices. The soundness of the model has been proved, that is especially liveness-, soundness-, and reachability properties that show the model's ability to fulfill essential networking functions. Furthermore the existence of energy optimal routes for fixed nodes are computed based on the energy requirements of MicaZ nodes. This analysis for example shows that higher transmission strength leads to a new optimal routing tree since nodes can transfer packets using less intermediate hops to deliver their packets to the sink.

Unfortunately all the analysis goes along with restrictions imposed on the complexity with different regards. As such the nodes' ability to do CSMA is only rudimentarily implemented since a more detailed model would no longer be suitable for analysis. In addition one has to confess that the restriction to small networks does not seem to be competitive with simulation tools. So the big advantage of the presented analysis should be kept in mind, that is the exhaustive treatment of the scenario which considers all possible states.

Within the focus of the current chapter is the energy required for the packet transport from the sensor nodes down to the base station. What we did not consider so far is the opposite direction. This is essentially useful to distribute information to the sensor network and is covered in the following chapter using an authenticated flooding protocol. Since sensor nodes may run out of energy, the effectiveness of the protocol is pinpointed for different parameter settings that directly correspond with the energy draw and the involved authenticity of packets.

CHAPTER 4

Performance Evaluation of Probabilistic Flooding Protocols

4.1 Introduction

Communication in wireless sensor networks is characterized by the fact that no centralized knowledge about the identity, reachability, or the location of the individual nodes is present. Although the nodes have been given an identity in terms of a MAC address or a unique ID before deployment, the knowledge about the topology after the initial setup is not available to every node. Thus it is important that nodes start to learn about their proximity, interchange routing information, and finally proceed with their task. The tradeoff hereby is whether information about the topology should be collected fast at the expense of energy or not.

An important class of security protocols depends on the use of probabilistic choices which are well suited for many purposes that also include signing contracts, sending certified email or protecting the anonymity of communication agents. Given these constraints a simple flooding strategy has become an accepted communication paradigm that was already topic in many previous studies and exists in many variations. Furthermore probabilistic flooding protocols play an important role in the context of low-energy wireless networks. Where in the simple flooding protocol every node sends the received message to all surrounding nodes, this happens in the context of the probabilistic algorithm with a likelihood $p < 1$. Here a node either forwards the message with probability p or drops the query with $1 - p$. In consequence this mechanism can greatly reduce traffic or packet collisions (*broadcasting storm*), and hence there can be significant energy savings. Broadcasting flooding protocols were intensely studied in the work of [NTCS99].

The here presented probabilistic analysis is executed by the use of the PRISM tool — a probabilistic model checker [KNP01, KNP07, HKNP06]. In particular its application to the field of wireless sensor networks is studied with respect to modeling, usability of the tools, and its suitability to the application domain. Although the tool might not immediately suggest itself for the job of analyzing probabilistic protocols that are more in focus of state-of-the-art simulation tools like NS2 [Net], a comparable analysis of model checking techniques can be conducted.

What made PRISM attractive from our point of view is its solid foundation on the theory of Markov chains and its comfortable graphical user interface. The size of the Markov chain models accessible to analysis by PRISM is restricted. This is partly caused by the fact that not only typical situations, that occur in 95% of all simulation runs are considered, but all possible interleavings are taken into account, no matter how low their likelihood to occur is.

Consider for example a topology as shown in Figure 4.1a. When the query is sent by a node A and each consecutive node propagates the query using a probability of 5%, the likelihood that the query reached node E is $6.25 \cdot 10^{-}6$ $(= 0.05^4)$. In turn this means that in 1 million simulation runs the query reaches state E on average 6 times. But it can also happen that the situation does not occur in 1 million runs. In fact about 100 million independent simulation runs are necessary to obtain meaningful results, which supports the idea of computing precise and accurate probabilities by the use of Markov chains.

Related Work Probabilistic flooding protocols are investigated in the work of [FG06] with respect to their general application. The authors focus on quantitative performance measures, compare the applicability of formal methods, and focus on quantitative performance measures. Moreover background information about probabilistic actions systems is provided. The work concludes by analyzing a system consisting of five nodes using a modeled channel behavior that accounts for clashes and transmission delays.

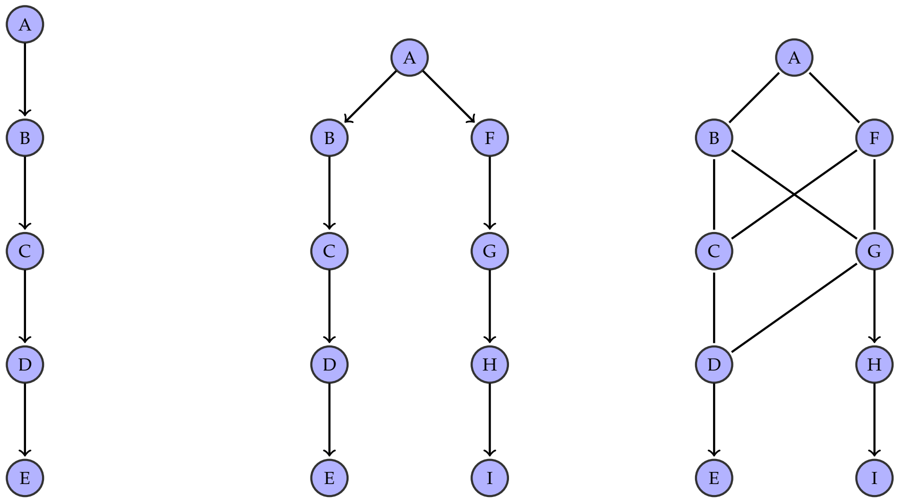
In the work of [KNS02] the authors apply probabilistic model checking to the IEEE 802.11 standard [Soc07] using a medium access control mechanism that employs CSMA/CA. The focus of this work is to investigate the randomized exponential back-off using a two-way handshake with a fixed network topology. A probabilistic timed automata model is then used to verify the quality of the back-off algorithm like "at most 5 000 microseconds pass before a station sends its query correctly".

In [KNSW04] symbolic model checking is analyzed with respect to its applicability to probabilistic timed automata. The authors present their symbolic model checking algorithms and apply it to case studies like the CSMA/CD protocol and the Fire Wire root contention protocol.

Overview This chapter is structured as follows: In Section 4.2 three simple topologies are analyzed on a theoretical basis with related question "What is the probability that node i receives a query?" Here also preparatory work is done that facilitates the development of PRISM model later on. In addition the PCTL query language, and communication mechanisms are investigated. Section 4.3 introduces the authenticated query flooding algorithm (AQF). The energy requirements are covered in Section 4.4 which also includes energy measurements of the TMote Sky Sensor node. Thereafter in Section 4.5 the results are presented for the relevant topologies. With the conclusion in Section 4.6 the results are discussed, analyzing the contribution and relating it to other work.

4.2 Probabilistic Flooding in Simple Network Topologies

In this section some simple settings are analyzed using a simple probabilistic flooding approach. It is of particular interest how such dissemination protocols can be transferred



(a) topology 1: Lined-up sensor nodes (b) topology 2: two-path divergence (c) topology 3: interlinked divergence

Figure 4.1: Basis network setups studied and modeled by Markov chains.

to Markov chains and to define the properties of interest. Since in the end a PRISM model needs to be constructed for the analysis of real world scenarios it is important to become familiar with the fundamental modeling concept. For this reason some simple settings are investigated and analyzed using the probabilistic flooding approach that are later expanded to more realistic network settings.

The simple topologies chosen are displayed in Figure 4.1, starting with a basic setting were the nodes are lined up in a row, over a more complicated example for which it is still possible to compute the resulting probability by hand. And finally a setting as shown in Figure 4.1c where different routes are possible for a packet to traverse. Furthermore what is different in this topology is that links change their nature from directional to bidirectional and hence the inter-connectivity of the whole setting is rising. Or stated otherwise, the arrows are used if there is only one possible way for the query to be sent through the topology. For the lines which interlink nodes, the packet can be sent in both directions depending on which node receives the query first.

The analysis using probability theory is not meant to have revolutionary character but rather tries to estimate the complexity of different topology settings and obtain a general "feeling" of how probability measures behave in such strongly interlinked structures and the way they can be computed. Later during this work problems are modeled in PRISM to compare the impact of different networking topologies w.r.t. energy considerations.

4.2.1 Networking Models based on Markov Chains

The PRISM model for the networks to be investigated is a DTMC. We do not consider an initial phase of the network where the topology is built up by exchanging routing tables to establish links. In this sense the topology is already fixed and not subject to changes. Each node is represented by a module having two internal states and communicate by the use of global bit variables which they can set to 1 to signal that a packet has arrived. So whenever a module sees a query through its global variable, the decision procedure is started by throwing a dice. Depending on the outcome it either forwards the packet with certain probability to its directly reachable neighbor nodes within one hop distance. Or the packet is dropped.

The probability that a node forwards a query to all neighboring nodes is referred to as the forwarding probability p_f. The related inverse namely the probability that packets are dropped and consequently not forwarded is denoted by $1 - p_f$. A node accepts a query if it forwards it, otherwise we say the node rejects the query. The node close to the base station which always receives the query is node A.

Topology 1 – Lined-up Nodes Consider a network topology where nodes are lined-up in row like in Figure 4.1a. We will start with this scenario to find a way for the PRISM models and then turn to more complex network structures. This scenario is relatively simple since the resulting probabilities can be computed by multiplying the probabilities of the individual events along the path. So, the probability that a node accepts and forwards a received packet can be computed by the use of basic probability theory as

$$\begin{aligned} &Pr[\text{query is accepted by exactly i nodes}] \\ &\quad = Pr[\text{query is forwarded by i-1 nodes}] \cdot Pr[\text{node i forwards the request}] \end{aligned} \tag{4.1}$$

The counter part is computed as the probability that there exist a path starting at the first node A from which a number of i nodes can be reached multiplied by the probability that node i drops the query. This can be formulated as the probability that node i discards a packet denoted as

$$\begin{aligned} &Pr[\text{request is dropped at node i}] \\ &\quad = Pr[\text{request is forwarded by i-1 nodes}] \cdot Pr[\text{node i drops request}] \end{aligned} \tag{4.2}$$

Further, if nodes are lined-up, it is obvious that node i can only receive a request if all the other $i - 1$ nodes forwarded it. Hence the probability that all nodes up to node i receive a request can be expressed by

$$Pr[\text{query is accepted by i nodes}] = p^i \tag{4.3}$$

Hence the probability that node i discards the query which all $i - 1$ nodes before accepted can be denoted as

$$\begin{aligned} &Pr[\text{request is dropped at node i}] \\ &\quad = Pr[\text{request is forwarded by i-1 nodes}] \cdot Pr[\text{node i drops request}] \\ &\quad = p^{i-1} \cdot (1 - p) \end{aligned} \tag{4.4}$$

In the simple scenario where the nodes are lined-up no 2 sensors can reject at a time since this would mean that nodes are skipped, in consequence probability

$$Pr[\text{request is dropped at node i and j}] = 0$$

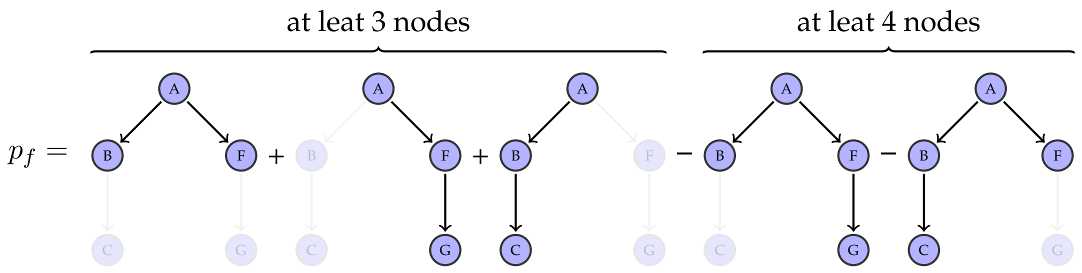


Figure 4.2: The way partial probabilities from problem 2 contribute to the resulting probability for the event, that 3 nodes accept the query.

.

In the PRISM model for the line-up scenario node A initiates flooding by running its probabilistic algorithm and proceeds with probability `p` setting the global reception bit of node B (`b'=1`) and hereby signaling that a packet arrived. Afterwards it is changing its local state to `a'=1` to remember that it has processed the query and to avoid multiple sending operations (packet repetition). With probability `1-p` it just moves to state `a'=2` and thus drops the packet. The PRISM sources are attached and can be found in Appendix B.1.

Topology 2 – 2-Path Divergence The topology of the two-path divergence as displayed in Figure 4.1b is treated in the following. In contrary to the afore mentioned scenario it is more sophisticated due to the divergence of the query into two distinct and independent paths after the first sensor node. As before the probabilities of events can be calculated by the probability of the event's path.

For example, consider the probability for three nodes being reached by a query in Figure 4.2. Basically three possible settings exist that represent this situation, namely B-A-F, A-F-G, and C-B-A. By simply adding the probabilities of the individual events, certain events are counted doubly. The paths that need to be subtracted can be constructed of the previous mentioned paths by finding the paths that only differ in one edge and unify their nodes. This can be done by subtracting all the possible combinations of events, having a predecessor and differ in exactly two nodes. The resulting formula for recursively computing the exact number of nodes that receive the request is as follows:

$$\begin{aligned}
&Pr[\text{query q is accepted by exactly i nodes}]\\
&\quad = Pr[\text{q is accepted by at least i nodes}] - Pr[\text{q is accepted by at least i+1 nodes}]\\
&\quad = \left(i \cdot p^i - (i-1) \cdot p^{i+1}\right) - \left((i+1) \cdot p^{i+1} - i \cdot p^{i+2}\right)\\
&\quad = \left(i \cdot p^i \cdot (1 - 2p + p^2)\right)\\
&\quad = i \cdot p^i (1-p)^2
\end{aligned} \tag{4.5}$$

At this point it turns out that the theoretical results computed by formula 4.5 and the outcome by PRISM do only perfectly match up to $i = 5$. Beyond that, some difference arises that is comparatively large at the beginning but then smooths for higher values of

i (see Table B.3.1). The reason lies in the modeling: In the PRISM model, a finite model is used with 9 nodes where the longest path is containing 5 nodes, that is A-B-C-D-E. Formula 4.5 in contrary anticipates an infinite model that is unbound on each of the two branches.

The figure for the number of nodes that drop a request is more complicated since even for $i = 1$ there are 20 possible paths that contribute to the probability, neglecting even the number of double-counted ones.

Topology 3 – Interlinked Divergence In problem 3 as displayed in Figure 4.1c, the scenario from before is extended by adding three additional links to the node pairs of B-G, C-F, and D-G. Hereby links in this setting change their function. For the first time some of the links obtain a bidirectional nature which was not the case in the previous settings and which dramatically increase the complexity of the model since many dependabilities need to be considered when computing the probabilities. In detail the number of transitions in the DTMC model grows from 172 in Topology 2 to 2 182 with 811 total states. Bidirectional in this context means that the communication is possible for two directions, in contrary to a one-way channel where one party can only send and the other only receive. Especially since all possible paths that reach a certain node need to be identified, it is almost impossible to solve this by hand. In summary, while topology 1 and 2 are easy to compute by hand topology 3 is not. Still, the probability can be computed by PRISM and is shown in the right illustration of Figure 4.5.

4.2.2 Inter Module Communication

The only communication is by global variables. This keeps the model relatively simple and reduces the complexity while essential functionality of the network links are still retained. As such the links are bidirectional and communication can take place in both directions. The communication is assumed to be reliable, meaning that whenever a packet is sent, it will reach the receiver with probability $p_c = 1$. When assuming communications below 1 to model lossy channels, the model can be adopted for this by incorporating the probability p_c into the decision of the probabilistic flooding algorithm. This would then mean in detail that instead of moving with probability p_f to another state one would move with probability $p_c \cdot p_f$ to that state and otherwise $(1 - p_c) \cdot p_f$ move to a "lossy state" where the packet is lost due to a communication failure.

It can also be explained from another point, that when considering regular networking conditions a BER (*bit error ratio*) of 10^{-6} is often used. Hence bit errors occur only once out of 1 million bits which can safely be neglected.

The fact that no collisions are modeled is motivated by the low forwarding probability p_f which is used throughout the model. In reality by doing simulation in NS2 it turned out that when dealing with probabilities p_c in the range of 10%, collisions are negligible since they almost never occur.

In Figure 4.3 the resulting model is displayed. It reads as follows: upon arrival at node A, the query is either dropped with probability $1 - p_f$ in which the Markov chain is stopping and the node changes to the sleeping state A_s. This helps for example to avoid duplicated request. With probability p_f the node decides to forward the request by moving to state A_t. Considering $p_c = 1$ as motivated earlier, node C and B receive the query and repeat

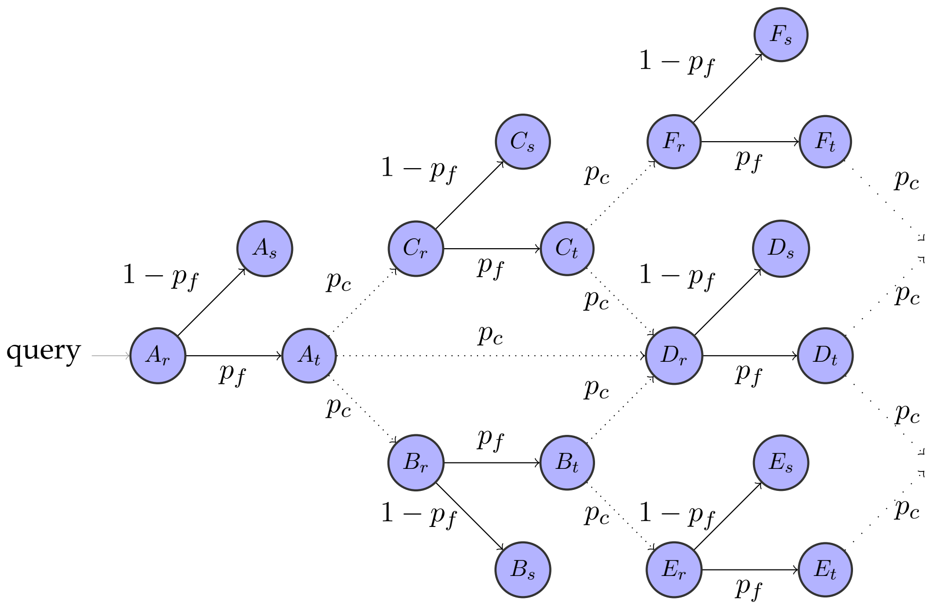
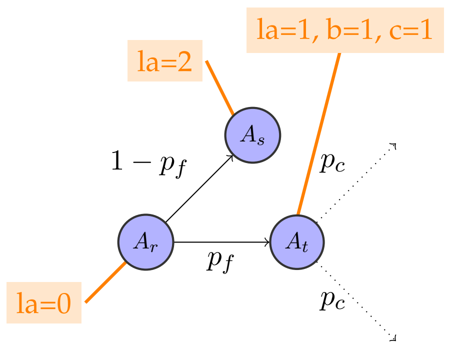


Figure 4.3: Communication model for the flooding protocol with p_f the forwarding probability and p_c the probability that a communication link fails.

the same procedure, but independently of each other. So this action is then repeated for all nodes until at some point either the whole network is reached, meaning that all PRISM modules are either in state A_s or A_t. When choosing the value of p_f too low, it can happen that not all nodes in the topology are reached, and the query is lost on its way through the network.

4.2.3 Specifying PCTL Properties

Before we start with the computation of the probabilities on the Markov chain model using PRISM, the appropriate specifications need to be defined in terms of PCTL formu-



(a) Model of node A with local and global variables.

```
//Global Variables
//0 - Nothing received
//1 - received and deciding
global a : [0..1] init 1;

module sensA
  //Local Variables
  //0 - Nothing received
  //1 - accepted query
  //2 - rejected query
  la: [0..2] init 0;

  [] (a=1) & (la=0) -> pf: (la'=1) & (b'=1) +
                       (1-pf): (la'=2);
endmodule
```

(b) PRISM model of node A.

Figure 4.4: Markov model and related PRISM description for a single node.

lae [BdA95, HJ94]. Afterwards the state space can be searched for a fulfilling state of the specification.

In particular it is interesting to know whether the whole network is reached given a certain forwarding probability or not. Additionally, labels are added which allow the user to reuse variable assertions for different property specifications. The following label named as ACC_j (see formula 4.6) becomes true if exactly j nodes accept a packet by setting their respective local variable to 1. The term l_i denotes the local variable l of module i. In addition if-then-else constructs are used in an abbreviation form as known from modern programming languages. For example $(c?a_1 : a_2)$ denotes `if c then a1 else a2`.

$$ACC_j : \left(\sum_{i=\{a,b,c,d,e\}} (l_i = 1?1 : 0) \right) = j \tag{4.6}$$

It turns to true if exactly j nodes change their local variables l_i to 1 meaning that they accept a query. The final PCTL property that returns the probability that j nodes accept the packet is then composed as

$$P =? \quad [true \quad U \quad (\text{“}ACC_j\text{”})] \tag{4.7}$$

Since the computed probabilities also comprise all intermediate stages, that is where the dissemination process is still in progress and the flooding did not terminate, the results are not very meaningful. Thus the formula needs to be expanded by a conjunction with the deadlock property. This property is a build-in construct in PRISM and turns true if the Markov chain is residing in a state where no further transitions are possible to take. When talking about the network model this construct allows only valid end states to be considered for the final probability where the flooding process terminates. So the probability that "exactly i nodes accept a query" upon termination follows by

$$P =? \quad [true \quad U \quad (\text{“}ACC_j\text{”}) \quad \& \quad (\text{“}deadlock\text{”})] \tag{4.8}$$

Similar to the accepting properties the rejecting properties are defined in PCTL. The label REJ_j added turns true if j nodes reside in a state where their local variable is set to 2.

$$REJ_j : \left(\sum_{i=\{a,b,c,d,e\}} (l_i = 2?1 : 0) \right) = j \tag{4.9}$$

As before the probability that "at least i nodes reject a request" is expressed through the following PCTL formula that already adopts for the "deadlock" state

$$P =? \quad [true \quad U \quad (\text{“}REJ_j\text{”}) \quad \& \quad (\text{“}deadlock\text{”})] \tag{4.10}$$

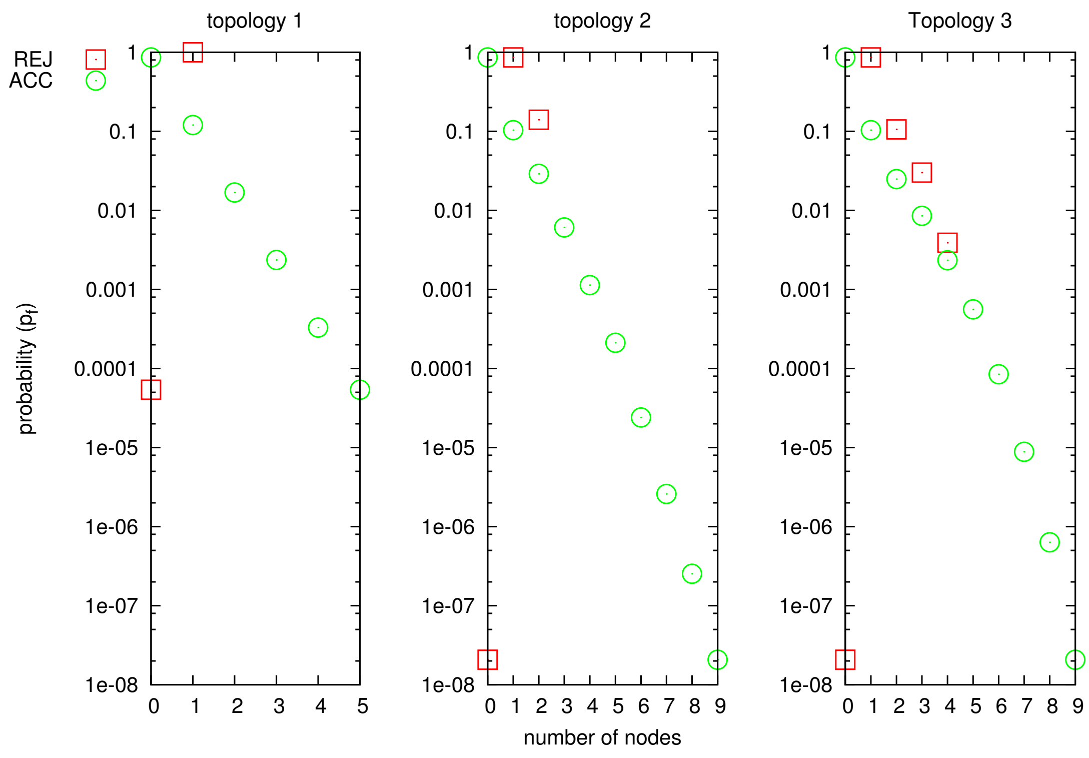


Figure 4.5: Probability for topologies 1 to 3 that exactly n nodes forward a query expressed through ACC_n and respectively drop a query denoted by REJ_n.

4.2.4 Results

All computations are carried out with the verification engine of PRISM unless stated otherwise. Therefore these results are exact since they precisely represent the probability as opposed to simulation where sometimes several thousand simulations runs are required to reach a desired level of confidence. In each topology the flooding algorithm always starts at node A.

The plot presented at Figure 4.5 shows the number of nodes being affected by a query in relation to the probability for the events "exactly i nodes reached" (ACC_i) and "exactly i nodes dropped the query" (REJ_i). For all experiments the forwarding probability is set to $p = 0.14$. The x-axis denotes the number of nodes. Due to the very quickly vanishing probabilities a logarithmic scaling is used on the ordinate.

For topology 1 (see Figure 4.5) there are only two outcomes for dropping the query. Either no node drops a packet or one does, but there can never be more than one node dropping a packet. All other cases only occur with probability 0. In topology 2 different scenarios for dropping nodes are thinkable, and in the final topology 3 there are 5 settings for dropping a query. In the last problem the probability for $REJ_{i\geq5}$ drops to 0 since the maximal number of rejecting nodes is 4. Similar, the maximal number of nodes accepting a query and forwarding it is 9 which results in $ACC_{i\geq10} = 2.07e - 8$

4.3 Authenticated Query Flooding Algorithm

Since sensor devices that interchange information through wireless communication are prone to attacks, there is an urgent need for security mechanisms. The challenge involved hereby is the fact that an adversary is hard to predict and often the attack stays undetected since appropriate detection mechanisms are missing. A comprehensive adversary model that is underlying this thesis is found in the foundation chapter at Section 2.4.2.

The number of thinkable attack scenarios and the adversaries potential cover a wide spectrum [BCF07]. They reach from a physical attack where the adversary is in possession of a node up to denial of service attacks where the nodes becomes inoperable. In case that the attacker brought a node under its physical control, it can read the node's memory including secret keys and reprogram it to suit to its needs. Or an adversary might post illegitimate or fake queries disrupting the networks service or compromise nodes in a way that let him control the nodes, or even parts of the network.

The *authenticated query flooding algorithm* (*AQF*) [BFH$^+$06] assumes an ID-based *key pre-distribution* scheme (see [ZXSJ03]). Out of a pool of keys numbered from 1 to ℓ every node receives a ring of k randomly chosen keys. The way this pre-distribution may be organized is sketched in [BFH$^+$06]. When the base station wants to disseminate a query q, it first computes the value $x = hash(q)$ of some given hash function $hash$ and then uses x as a seed to compute m pseudo random numbers $(kid_1, \ldots, kid_m)$. These numbers are interpreted as the numbers of keys $(k_{kid_1}, \ldots, k_{kid_m})$ from the pool. These keys are used to compute m *message authentication codes* (*MACs*). We stick to the design decision from [BFH$^+$06] to use 1-bit MACs. Thus, using key k_{kid_i} on $x = h(q)$ the bit m_i is computed. The sequence $(m_1, \ldots, m_k) = macs(q)$ is called the *authenticator* for q. The base station then floods query q together with $macs(q)$ into the sensor network.

Upon receiving a query q and the authenticator m a sensor node, which is assumed to share the hash function h and the pseudo number generator with the base station, computes the indices $(kid_1, \ldots, kid_m)$ used to encode m. If for at least one of the keys k_{kid_i} that also belong to its own key ring it detects a mismatch between the computed and the receives value m_i, it does not forward the query. In all other cases it sends it to all its neighbors. As in [BFH$^+$06], we are only interested in the analysis of the flooding algorithm and ignore the question when a sensor node considers a query as genuine and replies to it. Furthermore, the node memorizes processed queries and immediately ignores them when receiving them for a second time. It follows from the design of the algorithm, that obviously a legitimate query q will be received by all reachable nodes in the network.

The adversary model adopted in [BFH$^+$06] assumes that an attacker can feed messages plus authenticators into the network in the same way as the base station does. It is furthermore assumed that an adversary may *capture* sensor nodes and thus obtain their keys. Consequently it may start the dissemination with query q using the correct MAC-bits for the keys it has captured and randomly guesses the remaining authenticator bits. Assuming that an attacker has captured $\tilde{n}$ nodes using the theory of random sets the average number $\tilde{b}$ of keys known to the adversary and the expected value B of correct MAC-bits in a fake query authenticator can be computed.

The probability p_f that a sensor forwards a query with a fake authenticator [BFH$^+$06] can be derived through random set theory and is

variable	value range	description
n	12 14	number of nodes in the network
ℓ	1 000 - 10 000	number of keys in the key pool
k	50 - 250	number of keys in the key ring of a node
keylen	128	length of one key
m	100 - 500	size of the authenticator
MNKK	-	mean number of keys a sensor has to validate per query
data	8	data bits
d	2 - 4	network density
$\tilde{n}$	≥ 1	Number of captured nodes
$\tilde{b}$	-	number of captured keys
$E_{\tilde{b}}$	-	keys in the authenticator known by an adversary
B	-	number of right bits in the fake authenticator
p_f	-	probability that the message will be forwarded

Table 4.1: Annotation for the variable meanings and parameters for the AQF algorithm that contribute to the forwarding probability p_f. Blank fields depend on the setting and need individual computation.

$$\begin{aligned} p_f &= \left(\frac{\ell - k}{\ell} + \frac{k}{\ell}\frac{B}{m}\right)^m \\ &= \left(1 + \frac{k}{2\ell}\left(\frac{1}{m} - 1 - \left(1 - \frac{k}{l}\right)^{\tilde{n}}\right)\right)^m \end{aligned} \tag{4.11}$$

It is an essential contribution of the authors from AQF to suggest a criterion for choosing plausible values for p_f. Based on the theory of random graphs it is suggested to chose p_f in a way that the sensor network turns into a disconnected graph. This leads to the requirement that $p_f \cdot d < 1$, where d is the average number of neighbors in the network, also known as the density of the network. Table 4.1 shows typical parameter values that are investigated with the corresponding probability p_f.

Experiments considered later will use topologies with densities varying between 2.3 and 4.1 and forwarding probabilities between 5% and 40%. Especially when using smaller forwarding probabilities energy can be saved. This will satisfy the property of disconnected networks for forged queries due to the following fact: Fake queries are dropped with probability p_f but queries with a valid authenticator pass through the network with probability 1. So the disconnected graph only refers to a forged query.

Another property is that queries for which a node is not able to decide whether they are sound or not due to not-matching keys are forwarded by the node. An attacker could take advantage of this by sending queries with authenticator bits which only very few nodes know. Consequently these queries will be spread through the network draining the networks energy.

In fact, only a single matching key from the authenticator is sufficient for a node to accept it. This can very well be exploited by an adversary by just sending two queries, one with an authenticator set to 1 and the other to 0. This obvious vulnerability can be checked using a safety property. The safety criterion states that if a node s accepts a query (the query is legitimate), then with probability p_s it is legitimate.

Since another flooding — the simple authenticated query flooding (sAQF) — is checked for safety, and correctness properties, the task of verifying the safety of AQF is postponed to Chapter 5. Instead we focus here on energy related questions.

4.4 Incorporating Energy into Sensor Networks

Since the energy supply is a critical resource in the field of wireless sensors, this sections introduced six topologies. In combination with an energy model derived from the TMote Sky sensor node, the Markov chain model is analyzed. Especially in combination with the authenticated flooding mechanism, it is promising to find a tradeoff between energy and safety. Hence an optimal combination of the parameters can be computed which minimized the use of energy for a predefined safety level.

So far it seems evident to assume that average energy consumption of a node depends to a large degree on the sensor's proximity, since in dense networks the probability of occurring collisions or interference is much higher. Denser networks have the additional property that even when using small forwarding probabilities almost all nodes are reached.

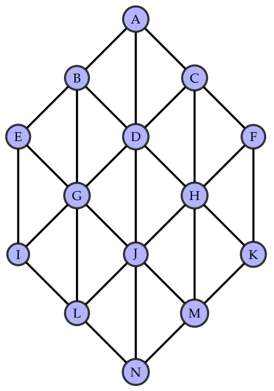
Where in Section 4.2 we did not differentiate between legitimate or fake query, this is changed in this section. From now on a faked query is considered which is injected into the network at node A although any other point is also feasible. Additionally the adversary can inject its queries at two or even more nodes in the network which is admissible, but not covered. In computing the energy balance from the network, all sensors will be taken into account to obtain a global energy view on the topology.

4.4.1 Application Scenarios

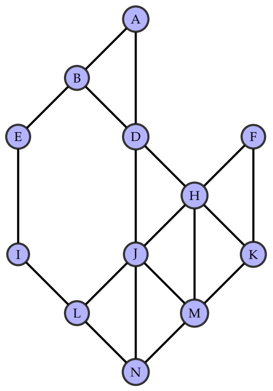
The networks that we will investigate are strongly interlinked, consisting of at most 18 nodes as shown in Figure 4.6 with nodes having at most 6 neighbors. The settings are chosen since they represent different symmetric and asymmetric topologies with a varying network density that will help to generalize the results. Furthermore topology 4 and 5 are similar and only differ in nodes G and C which are missing in topology 5. The setups in topology 6 and 7 have nearly the same number of nodes and differ only in their shape. Topology 8 has a tree-like look and queries that start at node A will later not interfere with each other since the branches are separated. The last topology 9 is chosen according to a real-world network deployed at Intel Berkeley Research [lab04]. In the subsequent part these topologies are analyzed under the objective of power constraints.

By modeling these wireless sensor networks as a DTMC (Discrete Time Markov Chain) the energy draw is computed using the reward analysis with the probabilistic model checker PRISM. So a sensor node specific energy consumption function is used to formulate the energy constraints within the model. The later analysis will reveal how parameters change when dealing with different topologies, and to which extent the impact of the topology will influence the level of security.

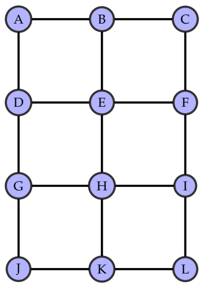
Notable here is the fact that no timing can be computed by the PRISM model. Although the timings for computing the hash values, receiving and sending for the TMote Sky node are present, the Markov Model does not allow timed modeling.



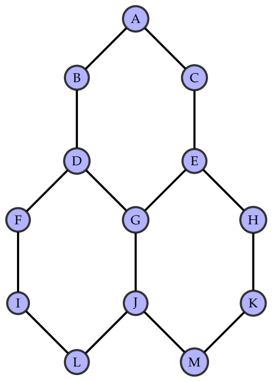
(a) topology 4



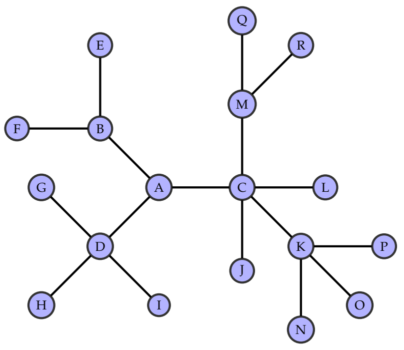
(b) topology 5



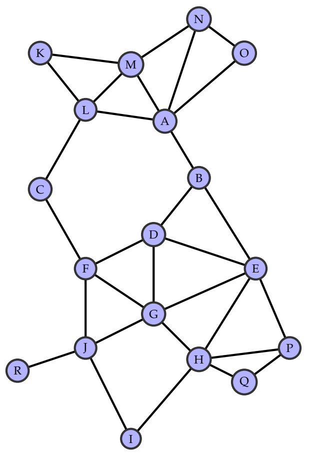
(c) topology 6



(d) topology 7



(e) topology 8



(f) topology 9

Figure 4.6: Considered symmetric and asymmetric topologies.

4.4.2 An Energy Model for the TMote Sky Node

For the energy model key figures from *TMote Sky* sensors by *MoteIV* [Mot06] are used and computed using a Perl script (see Appendix B.2). Hereby the exact energy for different packet sizes is exactly computed. The considered topologies, as mentioned before, are used for the analysis. The analysis shows how the choice of the topology is influencing the security-energy ratio.

The initial setup phase of the network during which routing tables and network parameters are negotiated are not considered since this routing information is in the network under analysis already present. More precisely, the connecting lines between the nodes represent these links. Especially the network's setup time and the hereby used energy does not only depend on congestion in the air but also on physical barriers like walls, trees, etc. that impede signal strength and the flow of queries. Since this may also vary with the topology it does not contribute to a comparison of the flooding algorithm in different settings. The here presented attempt is chosen to consider a network after successful termination of the initial phase since this greatly depends on the proximity.

Also noticeable is the fact that we do not account for state changes of the controller to obtain even more realistic results, since this influences the delay and the power constraints of sensor node, too. As such the here presented analysis gives a solution for a formally derived energy use.

Due to the fact that the TMote Sky sensor supports IEEE802.15.4 [Soc06, Zig05], no emerging packet collisions are considered since it is assumed that the underlying transport of packets occurs by time division multiple access (TDMA). As such, the here chosen approach is promising to deliver precise energy related data through formal methods, although real world results are expected to be higher since side effects are not considered in the model.

The energy operation figures that directly influence the reward model of the sensor network are briefly sketched in the following. For a detailed view of the background refer to the sources of the script at Appendix B.2. The energy draw of the radio controller integrated in the board is assumed to be $59mW$ for receiving, and $5.6mW$ for sending. The micro controller is able to compute 6 million operations per second (MIPS) with a related power need of $6mW$ (battery voltage of 3V). For moving data from the radio controller to the CPU and vice versa an energy amount of $65mW$ is needed, since the transceiver and CPU have to be switched on. All power needs are computed for a packet of 8bit size, a 128-bit key length, and increasing authenticator size.

Some energy figure with increasing authenticator m, related mean number of keys known ($MNKK$), and resulting energy of the TMote Sky Sensor board are displayed in Table 4.2. All sensors are either in receive mode during the flooding procedure, or try to authenticate the received query. Upon successful authentication, and in case of not being able to authenticate the data packet, they forward the query to nearby nodes according to the protocol description. Since propagation of fake queries should be limited to a small part of the network, variable p_f is chosen $< 1/d$. For some topologies, especially the dense ones, the forwarding probability need to be even lower.

			energy draw in $[mJ]$		
m	*MNKK*	p_f	E(R)	E(R+C)	E(R+C+S)
-	-	-	0.1648	0.1648	0.3280
100	1.67	0.439	0.2313	0.2574	0.4857
150	2.50	0.291	0.2645	0.2999	0.5608
200	3.33	0.193	0.2977	0.3425	0.6359
250	4.17	0.128	0.3309	0.3850	0.7110
300	5.00	0.085	0.3641	0.4276	0.7862
350	5.83	0.056	0.3973	0.4701	0.8613
400	6.67	0.037	0.4306	0.5127	0.9364
450	7.50	0.025	0.4638	0.5552	1.0115
500	8.33	0.016	0.4970	0.5978	1.0866

Table 4.2: Powers draw of the TMote Sky sensor node for the following operations: receiving E(R), computing and comparing the hash values E(R+C), and sending to E(R+C+S) for an 8 bit data packet and a 128 bit key length. The first line is without the AQF algorithm, for the remaining entries the total number of keys is fixed to $\ell = 6\,000$, the number of keys on each sensor is $k = 100$. Parameter $MNKK$ is representing the *mean number of keys known* by a sensor node and the authenticator parameter m is varying.

4.4.3 Reward Property

As input parameters for the PRISM model a 1-byte data packet is used. The key length for the authenticated query flooding is 128-bits, which seems to be a sound value considering an available memory of 8kB and the relatively high security level induced hereby. As varying parameters, the total number of keys in the keypool ℓ is chosen between $1\,000$ and $10\,000$ appropriate for small networks, the size of the authenticator m between 100 and 500, and the number of keys with which each sensor node is pre-loaded (k) to be in between 100 and 250.

The assumption underlying the model is that the adversary knows only the keys deployed on a single node, which are k valid keys. During the network operation, a node has to validate about *mean number of keys known* (*MNKK*) on average. This figure depends on variable input parameters and needs to be computed for each pair separately through

$$MNKK = \frac{m \cdot k}{\ell} \tag{4.12}$$

The *MD2* (*Message Digest Algorithm*) [Kal92] cryptographic hash function is used since it seems to be a good choice in the area of 8-bit micro controllers dealing with a key length of 128 bit. Using these numbers a varying forwarding probability p_f (that a sensor node accepts the query with a fake authenticator) is obtained. This probability is in turn used as an input parameter for the PRISM model. An example for varying the authenticator size m while ℓ and k are kept constant is shown in Table 4.2.

To compute the energy, the Markov chain model is augmented with information about the costs from the TMote Sky node using the *reachability reward* by the R operator. The reachability reward property [KNP07] associates a reward with each path that occurs in the model. More specifically, the total reward of the sum of *state rewards* for each state along the path plus the sum of *transition rewards* for each transition between these state is taken into account. To receive the energy that is computed by the Markov chain model, the following PCTL query is used. It considers the total reachability reward for the model, which is the rewards accumulated along a path until the solving state — in this case the "deadlock" state — is reached.

$$R=? \quad [F \quad (\text{``}deadlock\text{''})] \tag{4.13}$$

The term *"deadlock"* in the specification can be misleading, but it is defining the appropriate state within the model in which all queries are processed and no further model execution is possible. The convenient nature of this modeling is that it does not need any further specification. Whether the flooding algorithms stops, due to dropping of the queries or due to the fact that the whole network is already reached, does not need further specification.

4.5 Results

For a better comparison of the effectiveness for AQF, the energy of a *simply flooding* is computed in addition. Simple flooding is a protocol in which each sensor node receives and forwards with probability 1. Hence no hashes and authenticators need to be validated and each query is spread to surrounding sensor nodes.

At this point it is necessary to point out, that the results of the reward rate computation are not easy to compare since topologies vary in network density, the total number of nodes and results would show significant differences. To account for this, an average rate per node is computed. When considering these findings one should keep in mind the drawback of this figure. In fact the remaining energy of nodes is not evenly distributed over the network as sensor nodes close by the base station tend to have higher spending as others located further away. In the current setting it is particularly the nodes located close to the adversary that receive many forged queries during an attack but only forward a small portion to the network.

4.5.1 Comparing Energy Requirements of Networks

In the following experiment, it is assumed that only faked queries are sent into the network at sensor node A. So the energy draw is obtained for flooding a forged query and it can then be generalized for a certain percentage of faked packets. Figure 4.7 and 4.8 illustrates different topologies with varying parameter p_f denoted on the x-axis. Horizontal lines on top of the figure represent the power need of nodes without the authentication algorithm which is independent to the forwarding probability p_f.

The curves for the network topologies with varying parameters are left curved with an upward slope and each having a global energy minimum. These minima are topology dependent and vary with the forwarding probability p_f for a fake packet between 5% and 15%. Results are displayed in the appendix (see Table B.3.2) with the related authenticator size m and figure $MNKK$.

With increasing p_f, the level of security drops while more and more sensor nodes are affected by a faked query, and consequently the energy draw increases. Although the

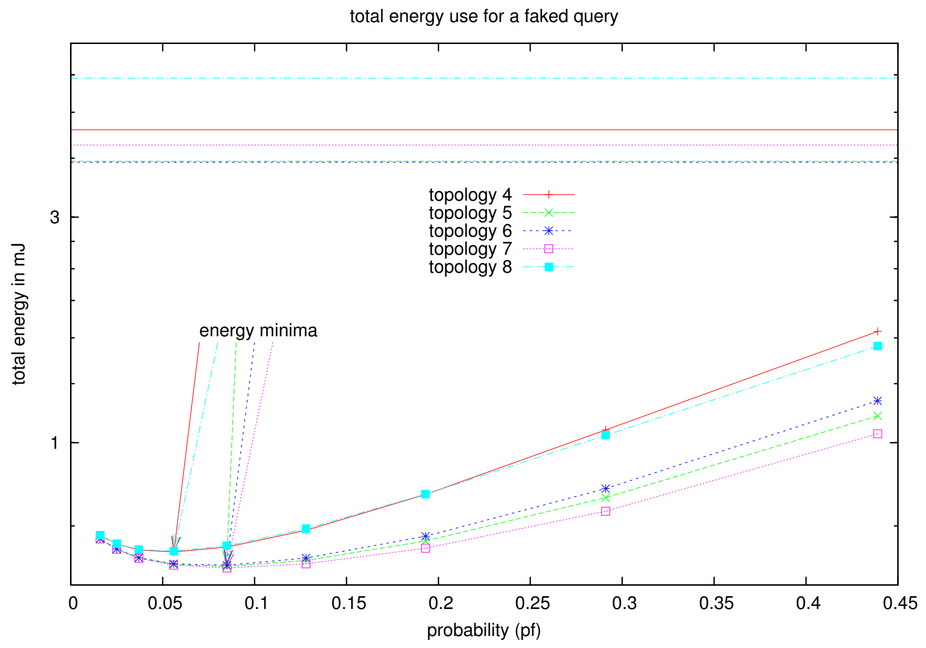


Figure 4.7: Total energy required for the flooding with varying AQF parameters shown by the curves. The horizontal lines indicate the energy level for simple flooding.

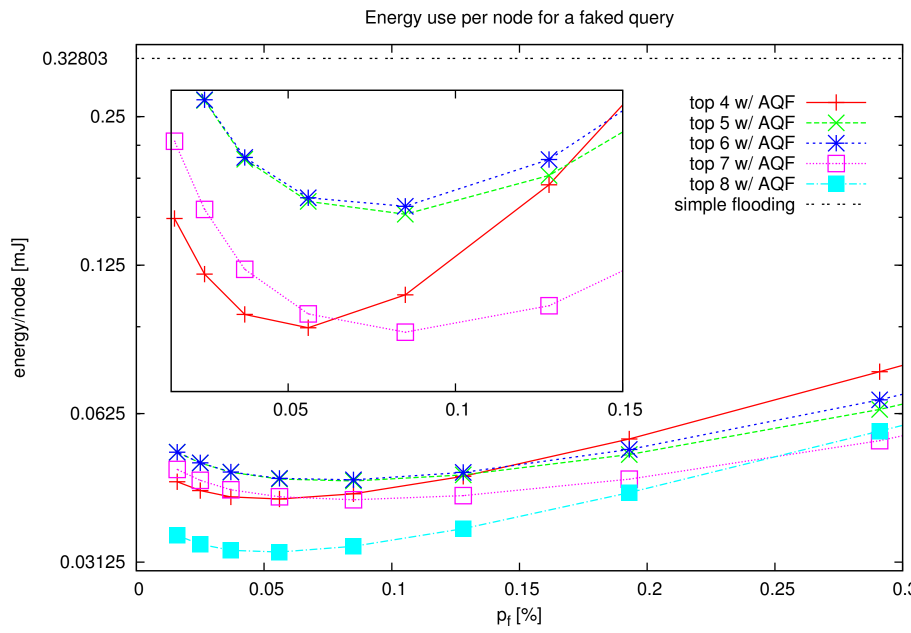


Figure 4.8: Comparison of energy requirements per node for different topologies using simple flooding and the AQF protocol.

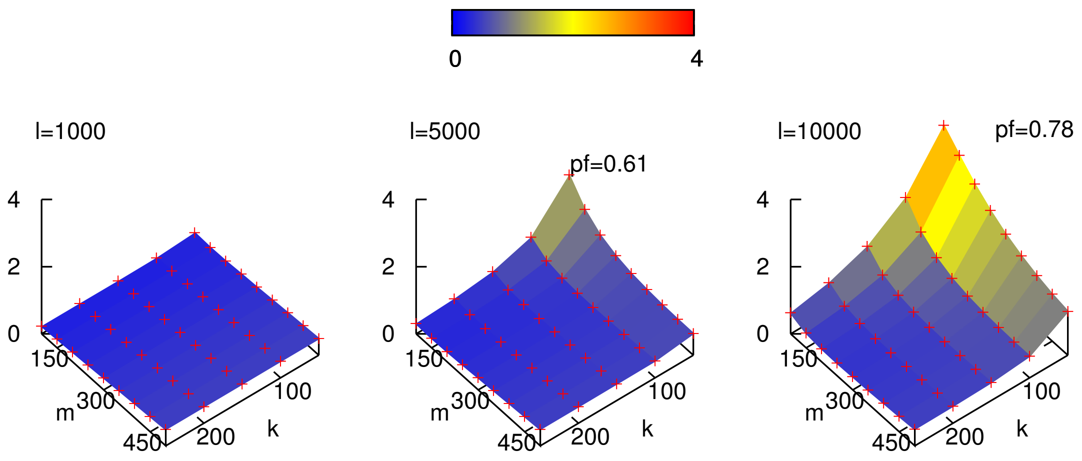


Figure 4.9: Input parameters like authenticator size m, number of keys per node k, and the number of keys in the keypool ℓ correlate with the energy use.

choice of topologies 4 and 5 are similar and only differ in 2 nodes, this has a visible impact on the power needs. Especially the missing node C in topology 5 reduces the connectivity at the first nodes, creating a "bottleneck" and flattening the flooding depth. The energy curve of topology 6 and 7 have a similar shape with a small offset. This is due to the fact that after node A the query is forwarded to 2 consecutive nodes and apparently this has an impact on overall consumed energy. Topology 7 has obviously the lowest energy draw for higher values of p_f thanks to the low interconnectivity between the nodes. According to this, its slope is comparatively flat, even up to a propagation probability of 40%. It turns out that for topology 9 no results can be computed due to memory restrictions, meaning that more than the available 32GB of memory are needed to successfully complete the model checking of this were not sufficient for the checking procedure. Although this scenario has only 18 nodes, the complexity is mainly caused by the amount of links.

As can be seen from the figure that although the additional computation effort is required by using AQF, the energy is far below the line of the unsecured networks when assuming that 100% of the queries are sent by an attacker.

Finding the right mix of parameters that determine probability p_f and especially the way they contribute to the energy is the key for successfully applying the AQF algorithm. Due to this task, Figure 4.9 illustrates their computed correlation with an average energy draw per node ratio denoted on the z-axis. It is remarkable that the three planes do have no intersection, meaning that there exists no single configuration that allows an optimal parameter setting for all scenarios. Instead, each network setup to be optimal needs a special configuration that includes also parameters like key pool size ℓ, authenticator size m, and the number of keys deployed on each node k.

Using a key pool with $\ell = 1\,000$ keys, the plane is relatively flat with an energy maximum at $k = 250$, $m = 500$ of $7.8598 mJ$. The value for p_f is at this point 0%, meaning that fake queries are dropped with a probability of 100%. With increasing parameter ℓ the plane shifts to a new energy maximum at $m = 100$, $k = 50$ of $17.9234 mJ$. Choosing a key pool of size $10\,000$ the power requirements even increase up to $28.6455 mJ$.

The explanation for this lies in the probability p_f for which we do only indirectly account through the parameter of m, k, and ℓ. In fact the evident increase in energy as shown in Figure 4.9 is due to the rapid increase of p_f which grows at many points beyond the admissible threshold of 50%, e.g., the energy maxima for $\ell = 5\,000$, $\ell = 10\,000$ are $p_f = 0.59\%$ and $p_f = 0.78\%$. For this reason it is important to chose parameter ℓ according to the topology since a keypool of $10\,000$ keys does not achieve the desired effect when having only 15 sensor nodes in a networking scenario. For detailed results refer to Table B.1 in the appendix.

4.5.2 Energy/Security Tradeoff using Topology 5

Since the energy efficient operation does highly depend on the level of security as well as the severity of the intrusion and the number of faked queries hereby sent, an energy-security correlation is required to phrase an objective statement. Otherwise no meaningful results can be obtained. Due to this reason the focus in the remainder of this section is, as motivated earlier in the introduction, to find an adequate level of energy that secures the network from outside intrusion. In particular, by estimating the number of nodes that an intruder brings under its control, the security level can be adopted to reduce the security overhead to a minimum.

By giving the solution for the energy-security tradeoff in relation to the number of queries sent by an adversary, the optimum operation parameters can be defined. Due to this reason a new variable is included into the model, which represents the ratio of fake queries among sound ones, ranging from 0 up to 100 percent. Since such a figure was missing in the previous setting, quantitative predictions can now be stated about how the energy and security level relate to the severeness of the intrusion.

This analysis considers the total amount of energy required to flood a query in topology 5 but can also be applied to other scenarios. For fixed parameters $\ell = 4\,000$, $k = 50$, and m varying between 100bit and 500bit, the resulting probability p_f and the hereby required energy is computed. At this point there may be more than one parameter configuration that lead exactly to the same forwarding probability, i.e., when doubling ℓ and k the same probability p_f is obtained due to obvious reasons. The only problem that shows up is that if the keypool size is doubled, the probability that a node has none of the keys required for the authentication of the query also doubles. Hence the keypool size has to be chosen appropriately to fit the number of nodes in the network. The corresponding Figure 4.10 shows the results.

The labels are percentage of faked queries on the x-axis, the security level as explained by the formula above on the y-axis, and the corresponding energy need in mJ on the z-axis. Two different data sources are contained in the illustration.

The red line on the left side at $p_f = 0$ shows the energy that would be used without the use of any securing mean. So no AQF algorithm is employed here. That is queries are received by a node and transmitted again without the computation of hash values, and validation of keys in between. In this case a constant amount of $3.94mJ$ is needed for dissemination of a query in the network which is completely independent from the probability of forwarding a fake packet p_f.

The second point of interest is the plane showing the relation of forged queries, and relate p_f to the amount of energy hereby involved. This graph reads as follows: If we assume only sound queries to be sent and only little security in use — that is p_f is high

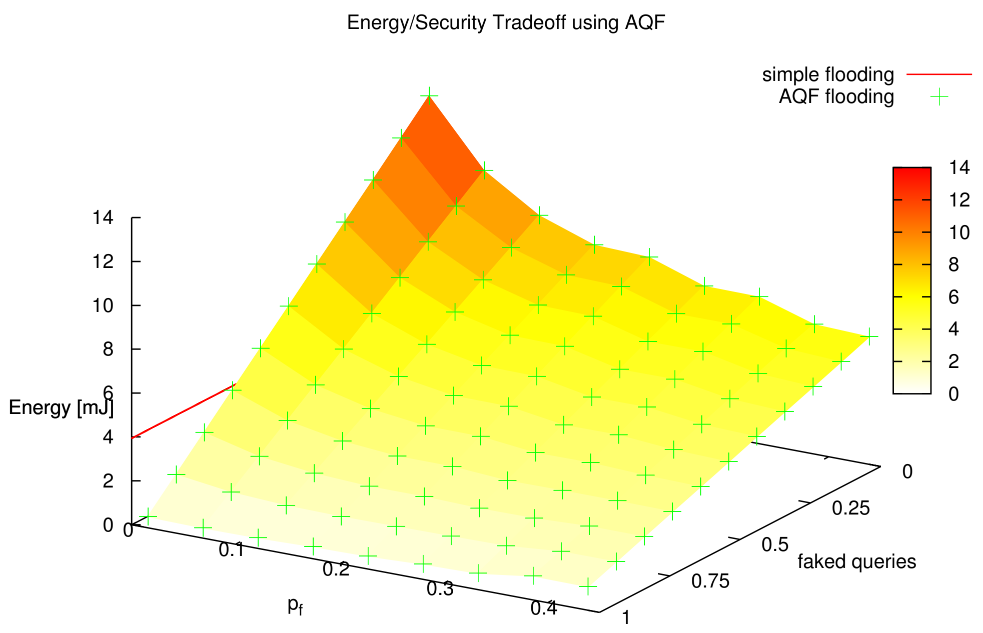


Figure 4.10: Area representing the relation between faked queries in percent, the probability of forwarding a fake query p_f, and the corresponding energy draw for topology 5.

— the AQF algorithm exceeds the regular flooding procedure with respect to energy. By increasing the portion of faked queries, the authenticated flooding shows effect and the energy draw starts to drop. On the other end of the scale ($p_f = 0.05$ and no fake queries) the situation is opposing, since the securing mean does not show any effect and reaches an energy maximum of $13.0392mJ$. As the number of faked queries that reach the network is increasing, more and more queries are "filtered" out of the network causing a rapid drop of the energy down to $0.5236mJ$. Still, the lowest energy use is not at the far end for $p_f = 0$, but at $p_f = 0.056$. The draw at this particular point is $0.4755mJ$ for a faked query but in contrary $10.3352mJ$ for a legitimate query. The AQF parameters used at this point are $m = 350$, $k = 50$, and $l = 3000$.

4.6 Conclusion

The analysis presented here shows how probabilistic model checking can be applied to challenging problems in the field of wireless sensor networks. Although common known problems like state space explosion prevail, the present application offers a wide range of investigation. Beyond the probabilistic probes that where conducted by PRISM, many other figures, mainly known from the field of network simulation tools, can be computed. As proven by previous work [WWB$^+$08], the application of model checking techniques is a well suited analysis method, having surely its drawback but also strong pros. Its strength and what differs them from common simulation tools is the high precision of the obtained results that go without the need of confidence levels since no deviation from the mean is present.

By using the cumulative reward functions quantitative assertions for a variety of properties can be verified. These include in detail the search for cost-optimal parameter settings. It turns out that although the number of nodes was far below hundred as proposed in [BFH$^+$06] still valuable results are obtained using formal methods. Essentially, the here presented analysis should be understood as a complementary approach that can be used to render simulation input parameters more precisely.

Hence it shows, against previous anticipations, that the choice of the topology has a tremendous impact on the network security, especially for probabilistic protocols like AQF. In particular the number of nodes that are in the neighborhood of the injected forged query play an important role. If they are sparse, the probability that the whole network is reached already drops below an acceptable threshold. The problem hereby is just that the attacked location is often unknown prior to deployment.

Though the analysis is restricted in the number of nodes, we could still compute forwarding probabilities with a high precision up to networks of 18 nodes. In particular it showed that not only the number of nodes contributes to the complexity of the considered networks. Also the interconnectivity of nodes is an issue since topology 8 and 9 have 18 nodes, but the forwarding probability of the later could not be computed due to memory restrictions. We further believe that the results presented here do not scale for bigger networks due to the program internal representation. Where in [CTTV04] the authors were able to decompose networks of homogeneous nodes ordered in a ring topology into verifiable components, this idea cannot be applied for the selected instances and the AQF algorithm. Fortunately since the forwarding probability is rapidly dropping to values below $p_f = 0.0625$, the probability that forged queries reach the inner network is dropping as well.

This chapter concludes the energy efficient analysis of the authenticated query flooding protocol. Up to this point we considered the energy draw of sensor nodes and tried to computed relevant energy draws by the application of formal methods. This happened in both directions. Once, the energy for packet sending from the nodes to the sink in the previous chapter, and in the current chapter, a probabilistic mean to broadcast information in an authentic way from the base station up to the leaves.

In the next chapters we change the focus from the energy related analysis methods and consider the verification and correctness of protocols and embedded software by formal methods. Up to now such properties only played a minor role. For this reason, we give formal proof, that the forwarding probability for a probabilistic flooding protocol is correctly expressed by a formula in the following chapter.

CHAPTER 5

Checking Formal Correctness of Probabilistic Query Dissemination

5.1 Introduction

The transport of information from the sink to the network is in the field of low power sensors often solved by the use of query dissemination. This method offers many means to achieve a desired level of reliability and security which can almost be arbitrarily chosen in combination with the power in use. The degree of freedom to design a protocol for energy efficient operation makes it well suited for applications especially in the field of wireless sensor networks.

Secure query dissemination is an often used scheme for the transport of information to legitimate nodes in a network. By the use of authenticated query flooding only an authorized entity like a base station will gain access to sensor nodes thus controlling the network by sending appropriate commands. Such an operation could for example be to instruct nodes within a particular region to start measuring temperature and forward these data values to the sink. By using authenticated dissemination a non-legitimate outsider should be hindered from spreading his own commands in the network, thus collecting for example private information or manipulate measured data values.

The authenticity of the query dissemination is induced by additional information called the *authenticator* that is appended to each packet sent through the network. This information is dynamically and independently computed for each query according to a secret combination of keys. So every node can check the authenticator of each query received and ascertain that the query is sent by a legitimate entity. Since only the base station is capable of computing an authenticator which is accepted by all nodes, an adversary has only little chance of breaking the authenticity of the protocol.

Most existing approaches that address similar problems have a deterministic nature meaning, that they allow an arbitrary sensor node to ascertain the authenticity of a query with a probability of one. So no mistakes in the validation of the authenticator are admissible. Since this high assurance level is costly to maintain, much energy can be saved when relaxing this energy consuming authenticator checking by introducing probabilistic algorithms. In more detail this means that a legitimate query is accepted by all sensor nodes with 100% probability. In contrast it is admissible that a node fails to discover a forged query sent by an adverse entity with a small but not negligible probability.

The authenticated query flooding is such a protocol introduced in the work of [BFH$^+$06] where the authors put special emphasis on large networks of the size of several thousand nodes. By relaxing the authenticity constraints from the AQF protocol it turns out that shorter authenticators would be much cheaper to compute. Since the number of sensor nodes is in practice in the range of ten up to twenty nodes [HCL08, BISV08], there is an urge to design new transport protocols appropriate for these tasks.

Due to the probabilistic nature of the algorithm simulations, conventional analysis strategies may fail to provide sufficient confidence. At this place the use of formal methods is an attractive solution since it establishes system and protocol correctness with mathematical rigor [Kat03]. Lately their success story gained popularity in the field of software as well as hardware verification. Due to the high potential involved they are suitable for the analysis of safety critical systems. Hence the application of those mathematical models for probabilistic algorithms is promising to deliver correct results due to their exceptionally high precision.

In this chapter the proposed *simple authenticated query flooding* (*sAQF*) protocol [Ben08] is analyzed using formal methods. Safety and liveness properties are considered that need to be fulfilled in order to guarantee proper operation of the protocol. By developing a probabilistic system that will comprise the formal model, a quantitative formula expressing the probability of accepting a forged query is derived and verified. In addition the outstanding check for the safety property for the AQF algorithm from Chapter 4 is provided.

In this way it turns out that existing simulations previously used to analyzed the protocol are not sufficient and will hence be supplemented by the use of formal method approach. By the help of Markov chains, probabilities can be computed with higher precision. Fortunately this will help to argue about the correctness of the derived formula as well as the authenticity and security of the proposed algorithm. Among the desired properties that should hold, criteria to assess the algorithm's safety and liveness are defined and validated against the formal model. Results will then be compared with deliverables resulting from simulation and the analytic formula.

Related Work Due to the fact that this work concentrated on the formal aspects of the proposed query dissemination protocol we will not descend to details of the protocol.

Probabilistic flooding protocols are investigated in the work of [FG06] with respect to their universal application. The authors focus on quantitative performance measures, compare the applicability of formal methods, and focus on quantitative performance measures. Moreover background information about probabilistic actions systems is provided. The work concludes by analyzing a system consisting of five nodes using a modeled channel behavior that accounts for clashes and transmission delays.

In previous work of [WS08] an earlier version called authenticated query flooding (AQF) was analyzed with quantitative performance figures e.g., the energy level of sensor nodes (see Chapter 4). Under various parameter settings topology dependent energy draws are computed that — as opposing to simulation — reflect the true performance of the protocol by precise and accurate values.

Overview This chapter is structured in the following way. Section 5.2 gives an introduction to the system model and the underlying simple authenticated query flooding algorithm (sAQF). The problem definition specified later motivates the use of the proposed method. In the subsequent Section 5.3 on page 74 a probability measure is derived using two distinct ways like *random-set theory* which is not accurate but therefore easy to compute. The correct derivation is based on *hyper-geometric distributions* (Section 5.3.2), deriving the correct probability but the computation requires extensive understanding.

The formal models for the correctness of the formula and safety property are explained in Section 5.4. In addition the safety property related to the AQF algorithm from Chapter 4 on page 47 is derived and checked. The results in Section 5.5 for the correctness proof show the differences between the two derived probability measures. In addition, the validity of the safety and liveness properties is provided for both query dissemination protocols AQF and sAQF. The chapter is concluded in Section 5.6.

5.2 Authenticated Query Dissemination

Before we start with a theoretical consideration about the protocol's properties the model and the algorithm's functionality are discussed. Later a simulation is given to illustrate one of the dissemination properties, that is illegitimate queries are dropped quickly and hence they only reach a small region of the network. Since these results are solely based on the forwarding probability and do not consider the algorithm behind the query flooding protocol, no knowledge about the query dissemination is necessary at this point.

Thereafter two different formulae are derived which both compute the probability of forwarding a forged query but using different approaches. The first solution is using Random Set Theory and the second a formula based on the evolution of the hyper-geometric distribution. The derived formulae and their results are formally verified for a specific parameter set and compared with the results from simulation.

The scenario that is used throughout this chapter represents a wireless sensor network consisting of n devices that fulfill a common task. The network is accessed by the user using the base station through which legitimate queries can be disseminated to the network. When using the authenticated query flooding (AQF) from Section 4.3 on page 56 queries can be authenticated by nodes using a key ring. Since none of the sensor devices does know all keys and the keys for the authenticator are randomly chosen, there is a chance that some of the nodes cannot authenticate the query. This is a disadvantage of the previous protocol which will be eliminated with the new proposed simple authenticated query flooding (sAQF) protocol.

The adversary model used in this work is firstly introduced in the foundation 2.4.2 on page 23 and did not play a role in the chapters before. It assumes that the adverse entity is in possession of several keys that he gained by capturing $\tilde{n}$ of n total sensor nodes. In consequence this means that the adversary can read the devices' memory including all cryptographic keys. The adversary can use these captured keys to inject forged and illegitimate queries into the network. If a node is not compromised the adverse entity has no means to figure out its cryptographic keys.

5.2.1 The sAQF Algorithm

The sAQF algorithm is proposed in the work of [Ben08] and works as follows. In the initialization phase a hash function $h()$ which does not need to be cryptographically secure is chosen. This function is known to all participants of the system and used throughout the protocol. The base station is loaded which ℓ randomly generated keys $key_1, \ldots, key_\ell$ with respective IDs $1, \ldots, \ell$ denoted as the *key pool*. Before deployment each sensor node receives its *key ring* consisting of k keys from the key pool. The key rings are randomly chosen and drawn independently for each sensor node according to a uniform probability distribution. In addition, each sensor knows the ID i that corresponds to the appropriate key in its keyring.

Before the query Q is spread by the base station, an authenticator is computed by hashing the query and obtaining $h(Q)$. For each key of the key pool $key_1, \ldots, key_\ell$ an *1-bit-MAC* b_i is computed by taking the last bit of the hash of the concatenation of key_i and the hash of the query $h(Q)$. The subsequent formula illustrates this computation:

$$b_i = \text{1\_bit\_MAC}(key_i, h(Q)) = [h(key_i @ h(Q))]_{last\_bit} \tag{5.1}$$

The message m is then constructed as a combination of the authenticator consisting of ℓ 1-bit-MACs and the query Q and can then be sent to the network as

$$m_Q = (b_1, \ldots, b_\ell) @ Q \tag{5.2}$$

What each sensor node does upon receiving the message m_Q is the following. It computes the hash of the received query using $h(Q)$. For all k keys key_i of its key ring a verification 1-bit-MAC b'_i is computed according to the formula (5.1). If for all computable b'_is it holds that $b_i = b'_i$ then the query Q is legitimate and originated from the base station. Otherwise the query is forged by an adversary and should be rejected. Doing so there are bits in the authenticator that the node cannot compute due to incomplete knowledge. These bits are simply ignored.

Essentially the main difference with the AQF algorithm from before is that instead of using m random keys for the authenticator, for the sAQF protocol the base station is using all ℓ keys from the key pool. In consequence where AQF uses a key pool of thousand and more keys, the sAQF only requires 200 to 400.

5.2.2 Problem Definition

The here defined issue is two fold. On the one hand a correctness proof for the quality of the dissemination protocol is necessary to argue about forwarding probabilities of queries. Only on the basis of a correctly derived formula, a proper evaluation of the forwarding probability can be obtained. The corresponding formula is derived in the sequel giving two alternative although mutual exclusive ways. Since the forwarding probability is considered to be a central and important part of the protocol this formula is verified using a formal model.

The second issue concerns protocol relevant safety and liveness properties that must naturally hold. To assure this, another model is created to test these validities. The following definitions are adopted from the work of [Ben08] and provide the problem definition of the authenticated query dissemination protocol sAQF. When dealing with energy related questions in Chapter 4, the proposed liveness and safety checks as promised there are still an open issue. Therefore we will also give a specification of these properties here and apply them to the *authenticated query flooding* protocol.

What makes the use of formal methods interesting in this particular application is the relaxation of the safety and liveness properties in a probabilistic way, namely to reduce the authenticator size and hereby contribute to energy savings. The relaxed definition looks as follows

Definition 3 (Probabilistic Authenticated Query Flooding) *A relaxed form of safety and liveness properties is defined by*

- *The property* ***probabilistic safety*** p_s *holds if a sensor node in a wireless sensor network (WSN) accepts the query* q *as legitimate, then with probability* p_s *it is a legitimate query.*
- *The* ***probabilistic liveness property*** p_ℓ *is fulfilled if any legitimate query* q *will be received by every node in the WSN with probability* p_ℓ*.*

In particular, the contrary position is often very useful that is talking about illegitimate queries. Doing so safety and liveness properties can be formulated in a similar way to reason about queries sent by an adverse entity. The properties from above can be adopted appropriately and read as follows:

Corollary 4 (Liveness and Safety for non-legitimate Queries) *We consider a query* q_{nl} *to be non-legitimate and* SN *a non-compromised sensor node within a* WSN*.*

- *The* ***non-legitimate probabilistic safety property*** p'_s *for a non-legitimate query reads as follows:* SN *treats with probability* p'_s *the query* q_{nl} *as legitimate and fails to detect it as fake.*
- *Similar the property for* ***non-legitimate probabilistic liveness*** p_ℓ *is fulfilled if* q_{nl} *will be received by every node in the* WSN *with probability* p'_ℓ*. Notice that* p'_ℓ *equals the forwarding probability* p_f *from Chapter 4 (*$p_f = p'_\ell$*).*

So far only the probabilistic versions of liveness and safety for non-legitimate queries are interesting to verify since their counter parts are already fulfilled by the design of the protocol. The properties for legitimate queries will hence be not treated.

5.2.3 Using Probabilistic Query Dissemination

In the following we assume that the mechanism how all the protocol parameters contribute to the AQF formula is known (see Formula (4.11) in Section 4.3). When adjusted to proper values a fake query would reach only a small region of the network, and fortunately this probability is even dropping very fast due to the multiplication of probabilities as shown in Section 4.2.4 on page 55. In order to illustrate this property a small topology of 14 nodes will be used as an application example as depicted in Figure 5.1. The topology is already introduced before in Figure 4.6 on page 59.

Underlying this scenario, the probability of forwarding an illegitimate query p_f is set to 30%. Within this topology a fake query is injected by the adversary at node E. Although the first node will very likely accept the illegitimate query, the probability is dropping when reaching the outer nodes located 2 hops away. So the probability for reaching node N with an illegitimate query is 12.826% whereas the probability for node K and node I is even 12.725%. The probability that one of these nodes even accepts a forged query is even less, namely 3.818%. Figure 5.1a shows the regions with similar likelihood of accepting a query.

5.3 Deriving a Probability Measure for sAQF

In the course of this section a probability measure is derived to compute the probability of an adversary to pose a query which will be accepted by a particular sensor node SN. The adversary is assumed to capture $\tilde{n}$ nodes each holding k keys. There is a pool P of keys containing elements $\{1, \ldots, \ell\}$. Each of the sensor nodes SN possesses a key ring of size k. In addition it is assumed that the adversary knows the key rings of $\tilde{n}$ captured sensor nodes which are at most $k \cdot \tilde{n}$ keys, but more likely less, since the key rings will have overlapping keys.

For the probability measure two different approaches are presented that will solve the purpose of describing the probability of a forged packet being accepted by a legitimate node. The first solution will derive a compact formulation based on Random Set Theory, neglecting a dependency relation between the parameters. But as can be shown, the results — especially those for scenarios with many captured nodes or very large key pools — will only marginally differ from the true expected probabilities. The second solution provides a recursive formulation with highly accurate results, that are complex to compute.

5.3.1 Random Set Theory

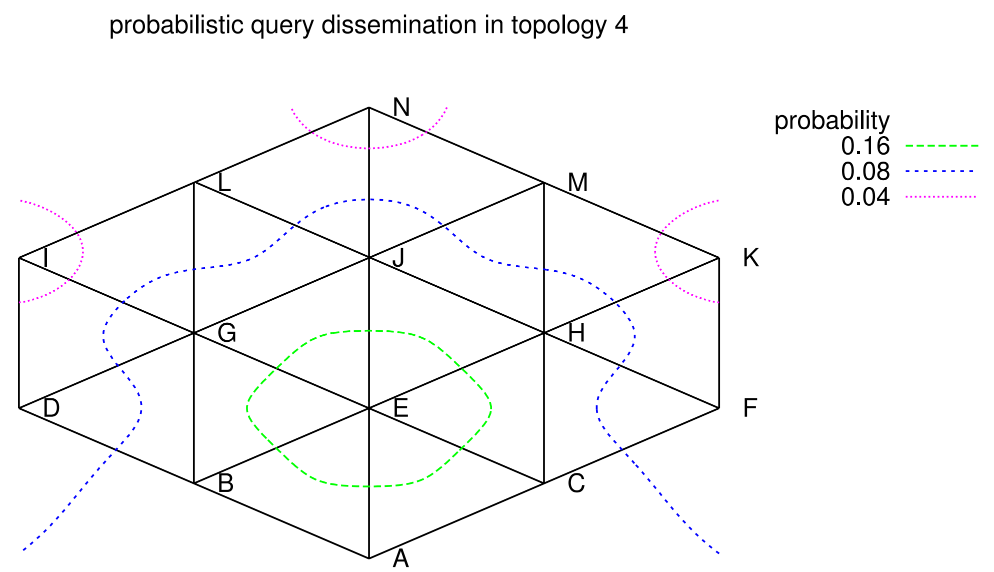
Every message m_Q that is flooded into the sensor network is accompanied with an authenticator — a sequence of ℓ bits which are computed using the sAQF algorithm — as mentioned before. The i-th bit in the authenticator sequence is computed using key number i from the pool P. When receiving a query each node checks k bits of the authenticator that correspond to the keys in its key ring. The adversary can read correct bits from $\tilde{n}$ captured nodes and guess the remaining authenticator bits. The probability that a forged message $\tilde{m}_q$ can be constructed in a way which makes it indistinguishable from a legitimate query to a sensor node is p_f. By legitimate query a message that originates from the base station is meant.

Let p^z denote the probability that the attacker knows exactly z keys of the key ring which should not be confused with an exponential expression. We obtain

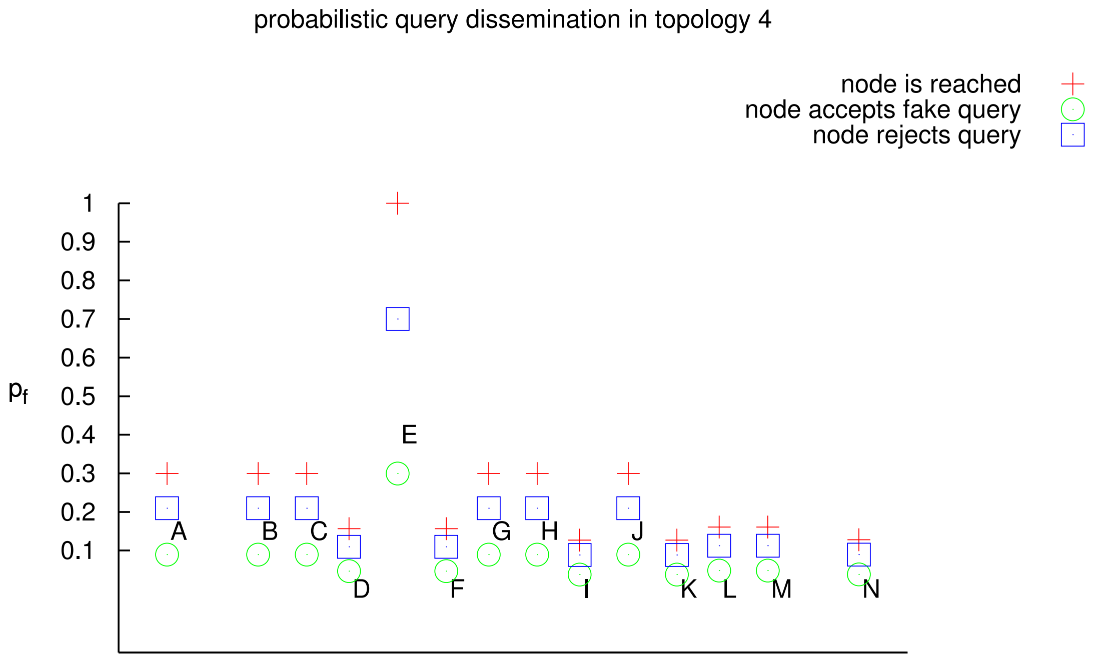
$$p_f = \sum_{z=0}^{k} p^z \cdot \left(\frac{1}{2}\right)^{k-z} \tag{5.3}$$

According to the results in Appendix C.1 the probability that the attacker does not know a key of *SN* is $(1 - \frac{k}{\ell})^{\tilde{n}}$. Thus the probability p^0 for 0 keys known is approximately

$$p^0 = \left(\left(1 - \frac{k}{\ell}\right)^{\tilde{n}}\right)^k \tag{5.4}$$



(a) Isolines showing the regions where a forged query is accepted by nodes with a certain probability using a node's forwarding probability p_f of 30%.



(b) Probability for each of the nodes of topology 4 to either *accept*, *reject*, or to be *reached* by a forged query.

Figure 5.1: Dissemination sequence of a fake query using $p_f = 0.3$

By approximately we denote that the above formulation of the probability is not fully correct. When using the formula from above we know that an unresolved dependence between the probability of knowing two distinct keys exists. For example if a node has one key ($k = 1$) and one node is captured ($\tilde{n} = 1$) then by the formula from above there is a non-zero probability that the attacker does not know any key. Or otherwise $k = l$. But since the formula that is derived using random-set theory is a compact formulation as will be shown later, we proceed with its derivation.

For a fixed key key_i in the key ring of sensor node SN the probability that key_1 is the only key that the attacker has captured is $(1-(1-\frac{k}{\ell})^{\tilde{n}}) \cdot ((1-\frac{k}{\ell})^{\tilde{n}})^{k-1}$. To keep formulas more readable we write $q_{unknown}$ or simply q_u for the probability that a fixed key is unknown to the adversary, and q_k (q_{known}) for the probability that a fixed key is known to the adversary. Thus $q_u = (1-\frac{k}{\ell})^{\tilde{n}}$ and $q_k = 1 - q_u$. So the probability that key_1 is the only key that the attacker has captured can be written as $q_k \cdot q_u^{k-1}$. Therefore, the probability that one key is known to the adversary is denoted as:

$$p^1 = k \cdot q_k \cdot q_u^{k-1} \tag{5.5}$$

The probability that two distinct keys key_1, key_2 in the key ring of a SN are exactly the keys known to the attacker is $q_k^2 \cdot q_u^{k-2}$. Summing over all two-element subset of keys we get

$$p^2 = \binom{k}{2} \cdot q_k^2 \cdot q_u^{k-2} \tag{5.6}$$

Now the general pattern is showing as:

$$p^z = \binom{k}{z} \cdot q_k^z \cdot q_u^{k-z} \tag{5.7}$$

Putting everything together we sum up over all possible combinations of known keys and obtain the probability:

$$\begin{aligned} p_f &= \sum_{z=0}^{k} \binom{k}{z} \cdot q_k^z \cdot q_u^{k-z} \left(\frac{1}{2}\right)^{k-z} \\ &= \left((1-q_u) + \frac{1}{2}q_u\right)^k \\ &= \left(1 - \frac{1}{2}q_u\right)^k \\ &= \left(1 - \frac{1}{2}\left(1-\frac{k}{\ell}\right)^{\tilde{n}}\right)^k \end{aligned} \tag{5.8}$$

5.3.2 Hyper-Geometric Distribution

We start our analysis by calculating with which probability the adversary acquires how many keys of the key-pool. We assume therefore, that the adversary captures nodes one

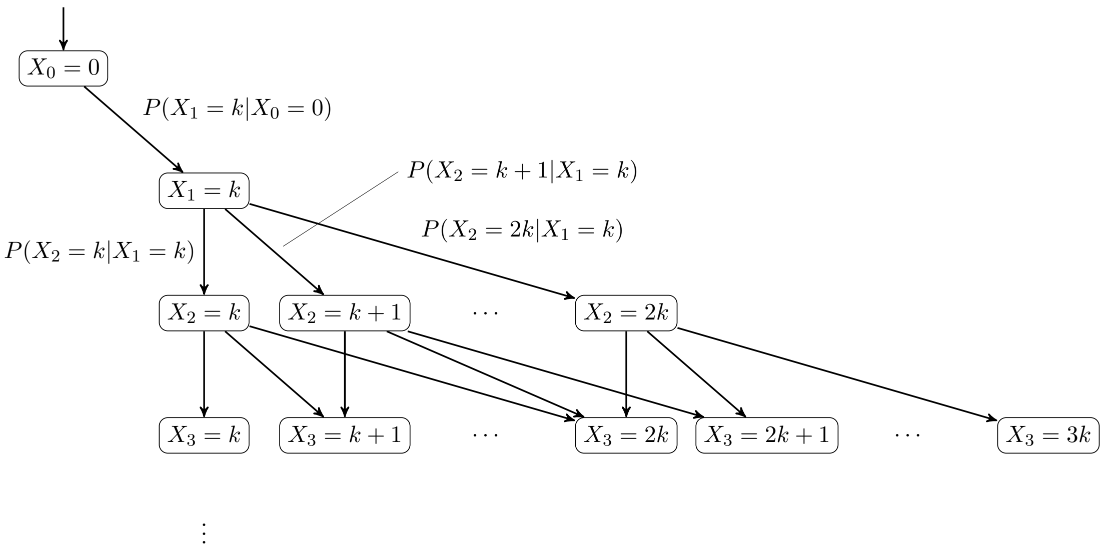


Figure 5.2: Probabilistic state model for the adversary where states for $X_i|i$ 3 are omitted.

by one, reads out their data and gains those keys (of the nodes key-ring) it did not have already before.

Afterwards we will be able to calculate the likelihood for a forged query, authenticated using the captured keys and guess the remaining bits, in order to formulate a query that is accepted by a sensor node.

As notation we write
$X_j :=$ number of keys known by an adversary after capturing j nodes
$X'_j := X_j - X_{j-1} =$ number of keys gained by an adversary on capturing the j-th node

The two parameters, number of captured nodes j and number of known keys a, in $X_j = a$ form a state of the adversary in a probabilistic model, shown in Figure 5.2. Transitions describe the capturing of one node and the keys gained hereby. There will always be $X_{j-1} = a \to X_j = a + b$ with $b \in \{0, .., k\}$ because on capturing one more node the adversary can gain 0 to k keys. Transitions are taken with certain probabilities $P(X_j = a + b|X_{j-1} = a) = P(X'_j = b|X_{j-1} = a)$. This enables us to assign probabilities $P(X_j = a)$ to the states.

In the beginning the adversary has neither captured nodes nor knowledge about keys ($X_0 = 0$). Expressed as probability:

$$P(X_0 = a) = \begin{cases} 1 & a = 0 \\ 0 & a \neq 0 \end{cases}$$

Obviously through capturing the first node he gains exactly the k keys known by that node ($X_1 = k$):

$$P(X_1 = a) = \begin{cases} 1 & a = k \\ 0 & a \neq k \end{cases}$$

This knowledge is never lost:

$$P(X_j = a) = 0 \text{ for } j > 1 \text{ and } a < k$$

In order to calculate probabilities of the other states we need to determine the transition probabilities $P(X'_j = b|X_{j-1} = a)$. It is only necessary to know how many keys the adversary already gained, but not which ones. This is because the key rings of different nodes are chosen independently and therefore the keys of the next captured node and those gained by the adversary from other nodes are independent too since we assume that no node is captured twice.

Capturing one node can be described using the *hyper-geometric distribution*: k keys are drawn at once (without replacement) of l total. $l - a$ of all keys are marked unknown to the adversary. Hence the probability of drawing b unknown keys is:

$$P(X'_j = b|X_{j-1} = a) = h(b|l, l-a, k) = \frac{\binom{l-a}{b}\binom{a}{k-b}}{\binom{l}{k}}$$

Putting together this formula and the restrictions to transitions we get
for $c \geq k$:

$$\begin{aligned} P(X_j = c) &= \sum_{a=c-k}^{c} P(X_j = c, X_{j-1}) \\ &\overset{X'_j = X_j - X_{j-1}}{=} \sum_{a=c-k}^{c} P(X'_j = c-a, X_{j-1} = a) \\ &= \sum_{a=c-k}^{c} P(X'_j = c-a|X_{j-1} = a)P(X_{j-1} = a) \end{aligned}$$

$\Rightarrow$

$$P(X_j = c) = \sum_{a=c-k}^{c} \frac{\binom{l-a}{c-a}\binom{a}{k-(c-a)}}{\binom{l}{k}} \cdot P(X_{j-1} = a) \tag{5.9}$$

Using Formula (5.9) we are able to calculate the probability distribution for the attacker knowing a certain number of keys. This is done starting with $j = 0$ captured nodes recursively up to $j = \tilde{n}$.
In a very similar way we develop a way to calculate the probability distribution about how many guessed authenticator bits a node is able to check, which in turn directly leads us to the probability of mistaking the query for a legitimate one.

Let Y be the number of keys known by checking node, but not known to adversary. Fortunately it turns out that $Y = X'_{\tilde{n}+1}$. That is because the keys known by the checking node but not by the adversary are just those the adversary could gain by capturing this one more node. Therefore

$$P(Y = b|X_{\tilde{n}} = a) = P(X'_{\tilde{n}+1} = b|X_{\tilde{n}} = a) = \frac{\binom{l-a}{b}\binom{a}{k-b}}{\binom{l}{k}}$$

$\Rightarrow$

$$P(Y=b) = \sum_{a=0}^{l} P(Y=b, X_{\tilde{n}}=a) = \sum_{a=0}^{l} P(Y=b|X_{\tilde{n}}=a)P(X_{\tilde{n}}=a)$$

$\Rightarrow$

$$P(Y=b) = \sum_{a=0}^{l} \frac{\binom{l-a}{b}\binom{a}{k-b}}{\binom{l}{k}} \cdot P(X_{\tilde{n}}=a) \tag{5.10}$$

Now, for every bit to be guessed and checked, there is a chance of $\frac{1}{2}$ of being right. This leads us to

$$P([\text{fake query accepted as legitimate}]|Y=b) = \left(\frac{1}{2}\right)^{b}$$

and hence

$$p_f = P([\text{fake query accepted as legitimate}]) = \sum_{b=0}^{k} \left(\frac{1}{2}\right)^{b} P(Y=b) \tag{5.11}$$

5.3.3 Differences in the Views

The two approaches tackle the problem from different perspectives and their advantages and respective drawbacks are evident. Using Random Set theory the results are overestimated using the Formula (5.8) as shown before since the dependence is neglected and especially for small numbers of $\tilde{n}$ the results differ most. Due to this reason the obtained results are higher than they should be. On the other hand this solution provides a way to compute the resulting probability using a compact formula, that is easy to compute.

The hyper-geometric Formula (5.11) is delivering precise outcomes at the cost of complexity. The implementation in a JAVA program shows the complexity of the algorithm mainly caused by the recursion and the required high internal high precision for internal variables (see Appendix C.3.3).

Preliminary sAQF Results

With the results from above a forwarding probability p_f can be obtained for different parameter settings. These results from the two formulas are compared with the output from the simulation script written in Python which is a contribution of Benenson et al. Doing so, the general tendency of the results obtained from the derived formulas can be recognized whether they roughly fit the simulation or heavily deviate from it. A comprehensive comparison between the two formulas, the simulation, and the verification is given later, after the formal model is introduced and evaluated.

Before starting with the verification work, different settings are simulated using the above mentioned script and the results are evaluated. As parameters the key ring size k, the key pool size ℓ, and the number of captured nodes denoted as $\tilde{n}$ are used. The

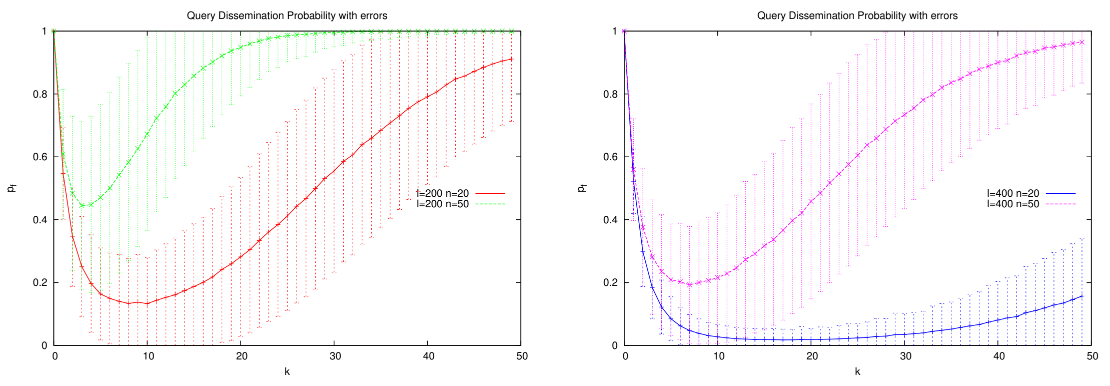


Figure 5.3: Simulation results with corresponding deviation around the mean for different parameter settings. Each value was simulated for 10 000 times.

simulation outcome is displayed in Figure 5.3 for 10 000 simulation runs per value with the variance around the mean indicated through the error bars.

What is noticeable is that the curves are not perfectly smooth due to the variance in the simulation results. This is a common known drawback when applying simulation. In the present scenario there is a high variation in the results. This is mainly due to the case that expected values expressed by the mean over all data samples does not represent the correct mean with a sufficiently high confidence. Rather the simulation results can be used as the best effort estimate. It is very likely that higher precision can be reached when increasing the number of simulation runs to some million since this would level out the outliers.

5.4 Proof Techniques Using Formal Methods

A formal proof using formal probabilistic verification is provided to ensure correctness aspects of the sAQF algorithm. By correctness it is first meant that the probability p_f that is obtained when applying formulas (5.8) and (5.11) equal the probabilities that originate from the formal model. This means that we only provide a partial verification since the range of input parameters is restricted to feasible values due to the limitations of the PRISM model. And if values for a fixed parameter setting equal the formal model, we consider them as correct, although in principle we should also give proof about the correctness of the formal verification model.

Second, the liveness property as stated before is checked whether it holds, or since the model is probabilistic this can be stated more precisely in a probabilistic sense as "What is the probability that a legitimate query is not accepted by a node or by all nodes in the *WSN*¿' Due to the different nature of both properties the second statement needs separate modeling.

In addition we are still in debt for giving a safety and liveness proof for the AQF algorithm from Section 4.3 on page 56. Luckily the same model as for the sAQF checking can be used to accomplish this.

5.4.1 Verification Models for the sAQF Formula

In the following a verification model is introduced as a Node and an adversary process based on the theory of probabilistic systems (see [McI06, BCHG99]) as introduced in the work of [BCHG99, McI06]. The model is used to verify the correctness of the simple authenticated query flooding algorithm.

Correctness of the sAQF Formula

The two processes modeled imitate the scenario where an adverse entity tries to construct an illegitimate query and send to an uncompromised node which will check the acceptance of the authenticator. Hereby the node- and the adversary process behave according to the protocol definition. After the model is completed, the verification is started by defining a PCTL query. The outcome of the verification results is afterwards matched with the two formulas for the *random-set theory* (see 5.3.1) and the *recursive hyper-geometric distribution* (see 5.3.2) on page 74. The comparison will then show whether the probability obtained when applying the formulas equals the results from the formal model.

Global System Definition The following global variables are used in addition to the probabilistic system description to analyze the model using PRISM. The following shorthands are introduced. We use for the assignment of an array index $n_i = 1$ as an abbreviation for setting the value of array n at position i to value 1, i.e., $n[i] = 1$. Moreover, we write $n_{0,\ldots,\ell-1} = 0$ to stress that all indices up to $\ell - 1$ of array n are set to 0, i.e., $n[0] = 0, n[1] = 0, \ldots, n[\ell - 1] = 0$. Arrays n, u, and a are read- and writable by all processes to establish a global inter-process communication. The parameters externally set by the user originate from the protocol. We use k as the number of keys per sensor node and $\tilde{n}$ to denote the number of captured keys by an adversary. For the length of the authenticator b denotes the number of bits to be checked. The final system is instantiated using the parallel composition without renaming as

$$System \mathrel{\hat{=}} Node \,||\, Adversary$$

Node Process The $Node$ process represents the sensor network device which receives a forged query from the adversary. Since the key distribution is random and independent for all sensor nodes this process is modeled as follows: The $Node$ process has variable $n_{0,\ldots,\ell-1}$ — an 1-bit-array of size ℓ — representing the key ring of one particular node. After the initialization phase is completed, the counter i is set to 0 and all k array values are assigned to 2. This value represents an undefined or unassigned value in the model. For the remainder the node process is choosing non-deterministically of its ℓ unassigned array locations k bits. When doing so a chosen array entry is set to either 0 or 1 with probability $\frac{1}{2}$. Subsequently the node holds k keys which are assigned to randomly chosen locations in the array as required by the protocol description. After this step the node process terminates. In terms of a probabilistic system the description of the node model is shown in Figure 5.4.

Adversary Process The process imitating the adversary behavior has two arrays (see Figure 5.5). That is an array $a_{0,\ldots,\ell-1}$ containing the keys already known and correctly

$$
Node \hat{=} \left(\begin{array}{l} \text{var } i : \{0, \ldots, k\}, n_{0,\ldots,\ell-1} : \{0,1,2\} \\ \text{initialy } i := 0,\ n_{0,\ldots,\ell-1} := 2 \\ []_{a:\{1,\ldots,b\}} : \ (i < k \wedge n_a = 2) \quad \rightarrow (n_a := 1 \wedge i := i+1)\,{}_{0.5}\!\oplus (n_i := 0 \wedge i := i+1) \end{array} \right)
$$

Figure 5.4: Probabilistic system of the node process.

$$
Adversary \hat{=}
$$

$$
\left(\begin{array}{ll} \text{var } j : \{0, \ldots, k\}, h : \{-1, \ldots, n\}, a_{0,\ldots,\ell-1} : \{0, \ldots, 3\}, u_{0,\ldots,\ell-1} : \{0, \ldots, 2\} & \\ \text{initialy } i := 0,\ a_{0,\ldots,\ell-1} := 2,\ u_{0,\ldots,\ell-1} := 0 & \\ []_{act:\{1,\ldots,b\}}\ (j = k \wedge h < n) & \rightarrow (u_{1,\ldots,b} := 0) \\ []_{act:\{1,\ldots,b\}}\ (j < k \wedge u_{act} = 0 \wedge a_{act} = 2 \wedge h < n) & \rightarrow (a_{act} := 3 \wedge u_{act} := 1 \wedge j := j+1) \\ & \quad {}_{0.5}\!\oplus (n_i := 0 \wedge u_{act} := 1 \wedge i := i+1) \\ []_{act:\{1,\ldots,b\}}\ (j < k \wedge u_{act} = 0 \wedge a_{act} \neq 2 \wedge h < n) & \rightarrow (u_{act} := 1 \wedge j := j+1) \\ []_{act:\{1,\ldots,b\}}\ (h = n \wedge j = 0 \wedge a_{act} = 2) & \rightarrow (a_{act} := 0\,{}_{0.5}\!\oplus a_{act} := 1) \end{array} \right)
$$

Figure 5.5: Probabilistic system of the adversary process.

probed and array $u_{0,\ldots,\ell-1}$ used to keep track of the captured bits in the current round. The values assignable to array $a_{0,\ldots,\ell-1}$ are 2 for *unknown bit*, 3 for *correct bit*, and 0 or 1 if the *bit is guessed* with the corresponding value.

Similar to the process from above, randomly chosen locations are assigned with value 3 in the adversary process to denote correct bits captured from a node. As such in the first round k bits that originate from legitimate nodes carrying a true keyring are assigned. By the use of array u (*unique bits per node*) we keep track that no bits are double assigned within one round. Not doing so would mean that a bit is read twice and in effect the overall results would be distorted since each node can only contribute once with each key. Whenever within one round a bit is used on a sensor it is indicated by setting the corresponding bit at array u. In addition we check by the guard $a_i \neq 2$ in line 5 whether the bit of the compromised node is already known to the adversary. In this case the bit does not need to be updated.

After all possible transitions are taken and consequently all k bits of the first compromised node are read, we proceed with the second captured node by incrementing the round counter h. In this round the same task is performed, namely filling up the bits an adversary knows from a legitimate sensor node. After k consecutive runs there is a high chance that some of the locations of array a are still empty ($a_i = 2$) since none of the captured nodes had these keys. For all of these locations bits are now randomly guessed, i.e., with probability 0.5 value 1 is assigned and otherwise value 0. If all unknown bits are guessed we end the adversary process and start with the probabilistic analysis.

PCTL Property Checking The following formula describes the probability of a faked query being accepted by a legitimate sensor node and returns a single numerical value. It states the probability to reach a state in which the properties expressed by the conjunction over all bits that contribute to the authenticator.

The conjunction term reads as follows:

If $a_i = 3$ so the adversary knows the right bit *true* is returned. Otherwise if $n_i = 2$ then the node does not know bit i and consequently it cannot tell whether the authenticator bit is correct or not. Last, if $a_i = n_i$ then adversary guessed the same bit that the node has.

So the node will falsely accept the query sent by the adversary and the probability to reach that state is returned. In addition the *deadlock* statement is added to stress that the desired state is final i.e., no further execution steps are possible. If all of the above failed, then the node discovers a forged authenticator.

$$P =?\, [\, true\, U\, (deadlock) \bigwedge_{i=1}^{bits} (a_i = 3\, ?\, true\ :\ (n_i = 2\, ?\, true\ :\ (a_i = n_i\, ?\, true\ :\ false)))\,] \tag{5.12}$$

Safety Property

To recall, the property as stated earlier is to verify whether an arbitrary node accepts a query q as legitimate. The reason for the safety property to hold is more than obvious. If we assume that the authenticator is composed of ℓ keys from the base station's key pool, of which each node's key ring is a subset, then a node can always verify exactly k authenticator bits. This property is trivially true and will always hold for legitimate query. In this case the probability is equal $p_s = 1$. So no further proof is necessary.

Liveness Property

This task is to check whether any legitimate query will be received by all nodes in the network. At this point it is worth to mention that the size of the authenticator appended to each query equals the size of the key pool. Due to this, the liveness property is evidently true.

In this trivial case each node shares at least one key with the base station and consequently a legitimate query having a true authenticator is reached by every node. Of course only assuming that the transportation layer of the network protocol does recover transmission errors. Only when considering flipping authenticator bits that stay undiscovered, a sound query that stems from the base station can be considered as fake. But since this case is treated by the reliable data transport layer, the liveness property is considered as valid and consequently needs no further treatment.

5.4.2 Safety of the AQF Protocol

When considering the authenticated query flooding protocol [Ben08] from Section 4.3 on page 56 the safety criterion is important to consider. That is the case if less than ℓ bits from the authenticator known to the node are readable. It can be the case that the authenticator from the base station does not even contain a single bit for which the node owns a key. In the following the authenticator m is set to be $m < \ell$.

$$Node \hat{=} \left(\begin{array}{lll} \text{var } i : \{0, \ldots, k\}, n_{1,\ldots,m} : \{0,1\} & & \\ \text{initialy } i := 0,\ n_{1,\ldots,m} := 0 & & \\ []_{a:\{1,\ldots,m\}}\ (i \leq k \wedge n_a = 0) & \rightarrow & (n_a = 1 \wedge i = i+1) \end{array} \right)$$

Figure 5.6: Node process for the safety property of the AQF protocol.

Scenarios with certain parameter combinations exist for key matchings that do not assure the 100% validity of this property. By the use of a formal model, this property is analyzed and verified in the following.

Global System Definition

Giving a model description by means of a probabilistic system, the following two processes represent the scenario under analysis, that are checked with PRISM. The variables defined are $n_{1,\ldots,m}$ to account for the key ring of the node and the appropriate counter part at the base station expressed through $b_{1,\ldots,m}$.

Variables exogenic defined are k to denote the number of keys per node, ℓ the size of the key pool and hence the maximum number of bits that can contribute to an authenticator. Finally variable m is fixed through the PRISM model. The final system is instantiated as a node and a base station using the parallel composition operator

$$System \hat{=} Node \parallel BaseStation$$

Node Process

The description of the sensor node process is shown in Figure 5.6 and does the following: after initializing the variables, k of its bits are set to 1 using the demonic non-deterministic choice. Hence, the key loading at the sensor nodes is imitated. Upon termination the process has exactly k key bits set.

Base Station Process

The algorithm implemented on the base station is similar to the one running at the node. After the initialization phase all bits that appear in the authenticator (m many) are set. As mentioned before we differentiate between ℓ representing the size of the key ring, and the size of the authenticator denoted as m. It is admissible to leave bits in the authenticator of a message blank, as in the example of the AQF algorithm, causing the safety probability to drop ($p_s < 1$).

PCTL Property Checking

The desired property is described by the following PCTL query computing the probability of reaching a valid end state in which property "*deadlock*" holds. In addition variable $CommonKeys$ is externally set representing the number of keys that are known to both node and base station. The terms in the sum are n_i to specify the value of the sensor

$$BaseStation \hat{=} \left(\begin{array}{l} \text{var}\; j : \{0, \ldots, \ell\}, b_{1,\ldots,m} : \{0,1\} \\ \text{initialy}\; j := 0,\; b_{1,\ldots,m} := 0 \\ []_{a:\{1,\ldots,m\}}\; (j \leq \ell \wedge b_a = 0) \quad \rightarrow \quad (b_a = 1 \wedge j = j+1) \end{array} \right)$$

Figure 5.7: Base station process for the safety property of the AQF protocol.

node bit at position i and b_i to denote the value at the base station at position i. What the query stresses is that either sensor node and base station have the same value at position i ($n_i = b_i$) in which case the value of the summand is b_i or it is 0. The term evaluates to true if the sum equals variable $CommonKeys$.

$$P =? \,[\, true\, U\, ("CommonKeys \;==\; \sum_{i=1}^{m} (n_i = b_i \,?\, b_i \;:\; 0)") \wedge ("deadlock")] \tag{5.13}$$

5.5 Results

Deliverables shown in this section are obtained by automatic evaluation of the probabilistic systems. For the evaluation of the models the PRISM tool (see Section 2.3.3 on page 17) is used with standard settings.

Since the capability of modern probabilistic verification tools is somewhat restricted in the number of states and the complexity, the verification tasks is executed using a keypool size of 8 bits. The simulation is adopted and fixed to the same parameters, using 10 000 simulation runs per value.

5.5.1 Correctness of the sAQF Formula

The results of the formal model are compared against the theoretical formulas from Section 5.3 and against simulation results. The number of bits used in the PRISM tool is set to 8 while varying $\tilde{n}$ between 0 and 8. For a parameter $0 \leq k \leq 8$ series are computed and plotted in Figure 5.8. The simulation results are obtained by the use of the Python script, the verification data is computed by the PRISM tool.

It turns out, that the verification results equal the computed probabilities using the hyper-geometric formula. Hence, Formula (5.11) correctly computes the probability of forwarding a fake query, for the used scope if input parameters. Beyond that range we have strong indication that the formula is still correct although this is not formally proven due to the limitation of the verification tool PRISM. For a choice of parameters out of that range, the simulation results fit perfectly with the computed probabilities of the hyper-geometric formula.

The results also show, that simulation and verification are roughly equal when neglecting the variance that is introduced by simulations. Since simulation experiments are repeated sufficiently often, their mean very closely fits with the probabilistic model checking data. What is obvious is that these results do not map to the results obtained when computing the random-set theory formula (see Figure (5.8)). Considering the figure for $\tilde{n} \geq 1$ there is a discrepancy between the correct value and the formula. At the

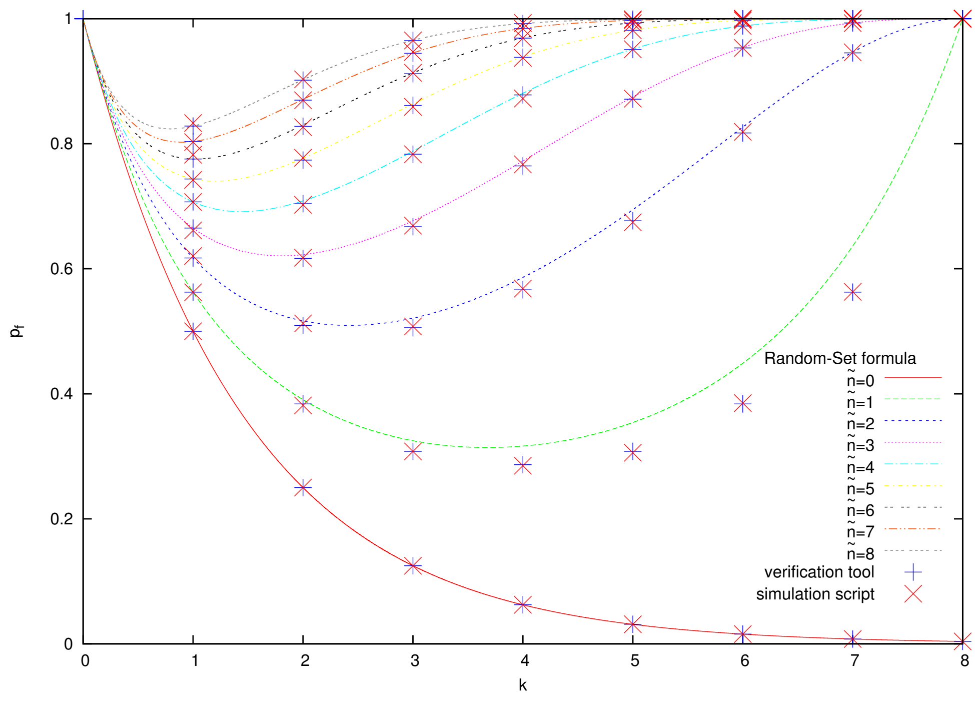


Figure 5.8: Comparison of results obtained by Random-Set-theory formula, the simulation script, and the verification with PRISM using ℓ=8. The values computed by the hypergeometric formula are omitted since they equal the verification tool results.

beginning ($\tilde{n} = 1$) this fault is relatively high but vanishes for higher values of $\tilde{n}$. Also noteworthy is that for values of $k = 1$ the failure does not show up since there is no dependency for a single key to consider. In consequence the exponent which contributed to the error disappears and the computed probability using the random-set formula is correct. Remember, this error is introduced by neglecting the dependency relation during the derivation of the formula as denoted earlier. We are aware of the failure but by the use of the formula a compact way of computing the forwarding probability is obtained at the cost of precision.

5.5.2 Restricted Safety Property for AQF

In order to argue about the safety property six different settings with a changing number of common keys are analyzed and printed. By *common keys* the number of keys is meant, known to the base station, chosen for the authenticator and in addition they are also known to a sensor node. This is in fact important for the following reason: If base station and a node share no keys, the query cannot be authenticated and its legitimacy cannot be assured. In this case the query is sent to other nodes for authentication. On the other hand using only few keys in the authenticator weakens the authentication mechanism since an adversary can guess one bit with probability 0.5.

As denoted earlier, the authenticator size is smaller than the keypool size ($m < \ell$). The case that $m = \ell$ is treated in the simple authenticated query flooding (sAQF) in Chapter 5.2.1 on page 72, where the safety property becomes trivially true.

The assumptions underlying the results are as follows. Authenticator m is varying between 2 and 10 bit in steps of two bit. The keypool size ℓ is fixed to 16 bit and the number of key per node k varies between 1 and 8 keys. Higher values for k above 8 can be ignored since the AQF protocol uses a key ring size of at most one fourth the size of the key pool.

Although the input parameters are small in comparison to the proposed values from the AQF description, the results scale for larger numbers. Figure 5.9 shows the obtained results computed with PRISM. Figure 5.9a shows the probability that the authenticator of size m send from a base station with keypool size ℓ and a node with key ring size k share exactly zero keys. This is the bad case since the node cannot authenticate a legitimate authenticator.

For $m = 2$ this value is still considerably high, but is dropping for higher values of m and k. Essentially this is caused since the number of shared keys exceeds 1 as more and more keys are shared in the authenticator. As an example, for $k = 6$ and $m = 8$ the probability is vanishing small around 0.35% that no common keys are shared.

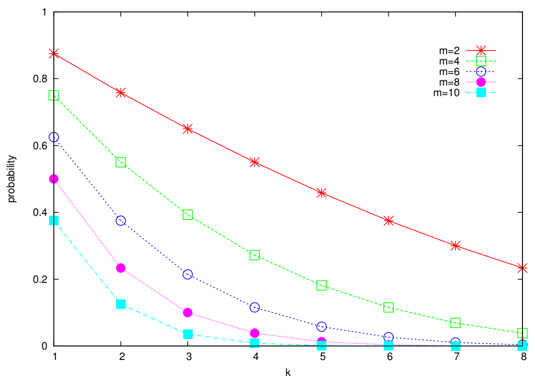
In Figure 5.9b where nodes know at least one key from the authenticator, the probabilities for $m \geq 6$ and $k \geq 2$ are above 0.5. Note that this figure also includes the probabilities for 2 and more keys that contribute to the final probability. This is still a bad configuration and only for higher values of m and k a probability above 95% is reached.

The remaining Figures 5.9c to 5.9e show probabilities for more common keys between an authenticator and a node. It shows that as Figure 5.9e the probabilities for the node to know four keys out of the key ring are relatively low. As an example consider the setting $m = 6, k = 5$: With a probability of 94.23% at least one key is shared, a minimum of two keys is shared with 65.38% and there is still a fifty percent chance that 3 keys are shared.

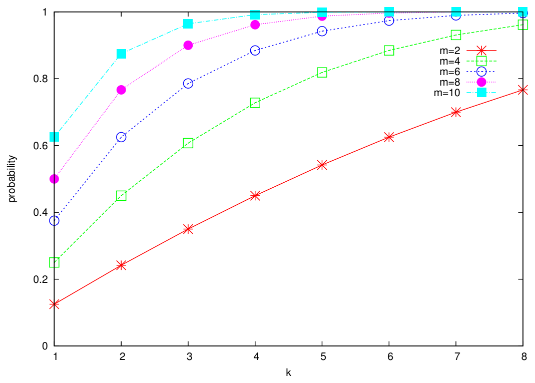
5.5.3 Forwarding Likelihood based on Hyper Geometric Distribution

The hyper-geometric formula that is derived in 5.3.2 is implemented in a JAVA program to compute a forwarding probability p_f based on different settings. The results are shown for two series with $\ell = 200$ and $\ell = 400$ in Figure 5.10. On the Y-axis the number of captured nodes $\tilde{n}$ is denoted and parameter k on the X-axis. As in the first setting (see Figure 5.10a) with a key pool of 200 keys the protocol can still handle up to 10 captured nodes independent of k. Of course the parameters of k have to reach some substantial value to introduce some security. But even values of $k = 7$ suffice to reduce the forwarding probability even up to 10 captured nodes below 5%. So using a parameter k between 7 and 13 provides a sufficiently low security close to 0. As such it is not surprising that the lowest probability is reached for small numbers of $\tilde{n} < 5$ and high $k > 45$ that are almost 0.

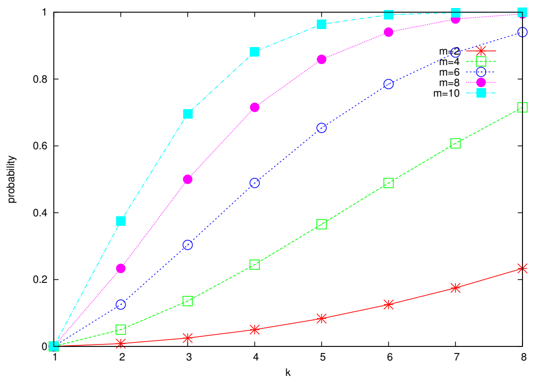
When using a key pool of $\ell = 400$ the situation is scaling in a sense that now twice as much captured nodes lead to the same probability than in the situation before. Now for values of k around 10 and $\tilde{n}$ up to 40 the authenticity induced by the sAQF algorithm even prevents that forged messages are sent with a probability around 3%. The figure clearly shows the danger of using to many keys per node in a situation with high values of $\tilde{n}$. Doing so raises the probability p_f rapidly to unacceptable high values that do not longer provide a decent security in the network.



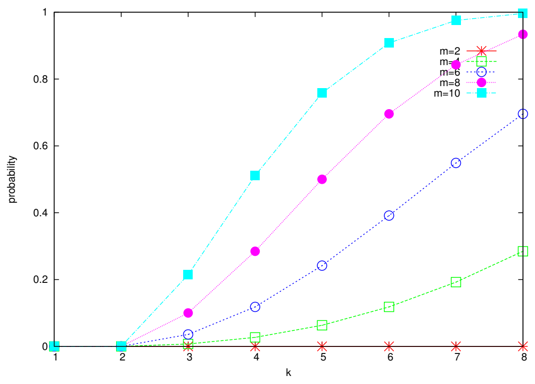
(a) node shares no key with base station



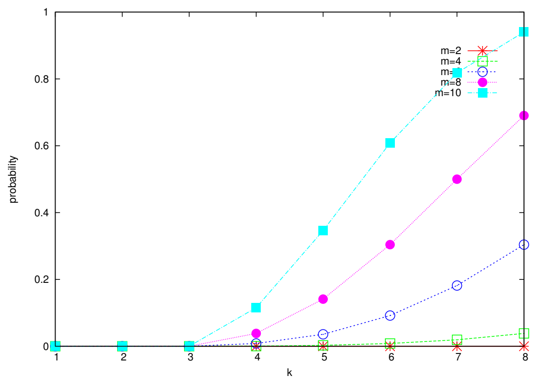
(b) probability $\geq$ 1 common key



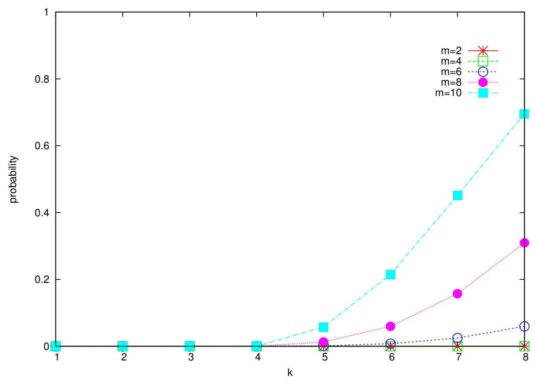
(c) probability $\geq$ 2 common key



(d) probability $\geq$ 3 common key

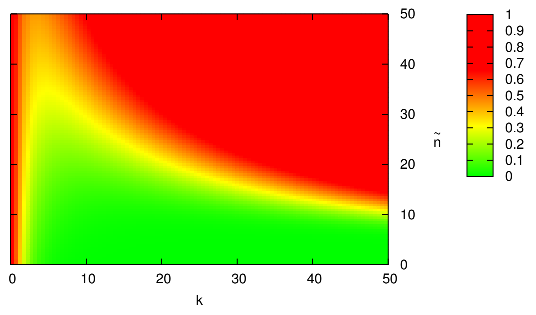


(e) probability $\geq$ 4 common key

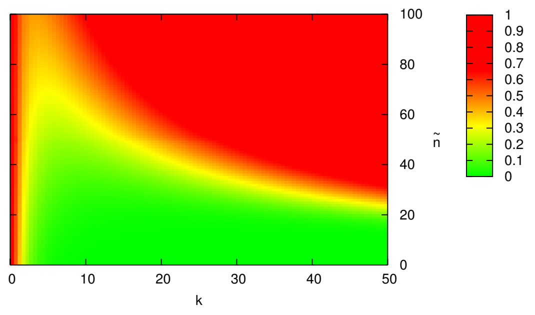


(f) probability $\geq$ 5 common key

Figure 5.9: Probabilities that node and authenticator from the base station share keys.



(a) sAQF with $\ell = 200$



(b) sAQF with $\ell = 400$

Figure 5.10: Probability of forwarding a forged query under certain parameter settings. The data is computed based on an implementation using the hyper geometric distribution.

When comparing the two illustrations on should keep in mind that there exist technical constraints on the maximum message size that can be sent. Using a ℓ of 200 means that the authenticator is also 200 bit. When switching now to $\ell = 400$ the security is increased but at the same instance the maximal size of a message content is reduced by $200bit$ $(25byte)$ since the authenticator required more space.

5.5.4 Simulation versus Verification

When comparing the results obtained by simulation the drawbacks become obvious. Simulated models generate data with a high variation which can be combat by sufficiently many simulation runs to reach results with an adequately high precision. In general this level of confidence is difficult to estimate when only doing simulation. In this case no reference values can be considered for comparison and the question "how much is enough?" is not easily answered. Presumably even when doing over 1 million runs, the results will still embody a strong variation. To illustrate this in more detail, error bars are added to the plot in Figure 5.3. What the error bars denote is the standard deviation of the simulated probabilities.

In addition consider Figure 5.11. The upper four plots of the graph show the results of the forwarding probability p_f when using the sAQF algorithm. This probabilities are computed by use of the hyper-geometric Formula (5.11) which computes exact results. The lower plot depicts the relative deviation in percent which is induced by the simulation.

Even for 10 000 simulation runs as in the present case the deviation is on average around 1%, in some cases even above. For example consider the setting of the blue line $(l = 400, \tilde{n} = 20)$. Here some outliers almost reach 4%, but on average they are less.

5.6 Conclusion

The present results show the importance of thoroughly exercised correctness proofs and quality estimates for query dissemination protocols. Especially the impact of their results and the vast range of application domains make them especially useful.

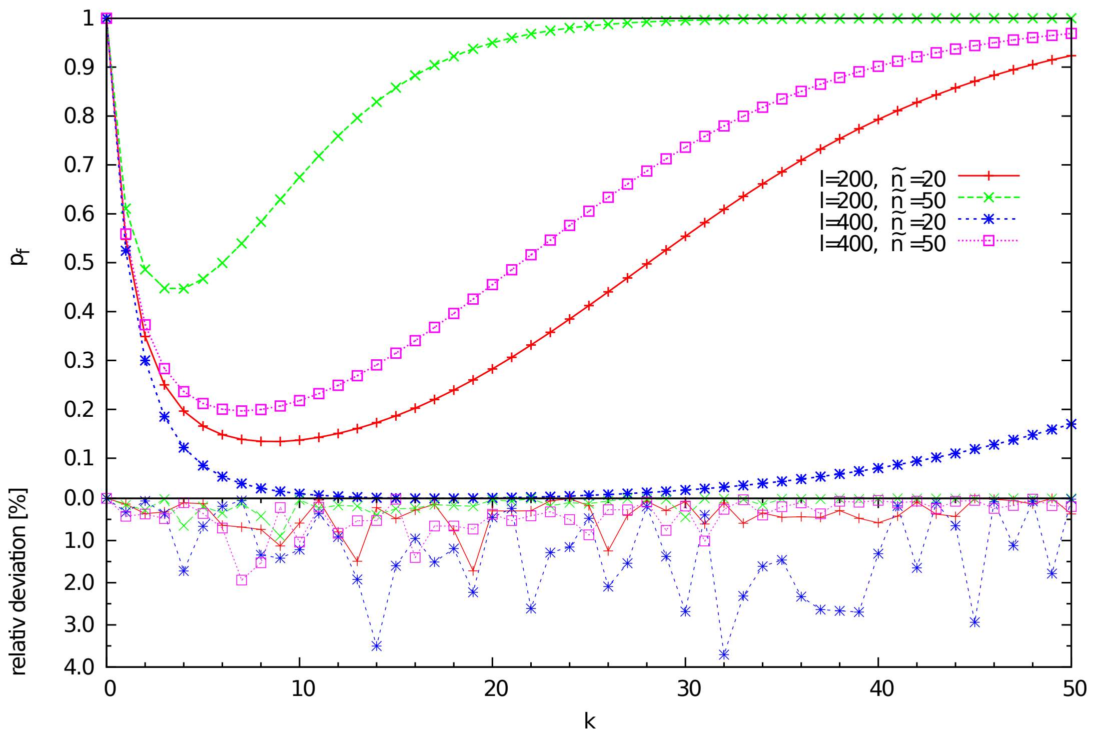


Figure 5.11: Computed values for the sAQF algorithm and the relative deviation of the simulated values for 10 000 runs.

		AQF	sAQF
legitimate queries	p_s	$(1-\epsilon) \leq$ (see 5.5.2)	=1 (see 5.4.1)
	p_ℓ	$(1-\epsilon) \leq$	=1 (see 5.4.1)
forged queries	p'_s	$p_{f\,AQF}$	$p_{f\,sAQF}$ (see 5.5.1)
	p'_ℓ	$< \epsilon$	$< \epsilon$

Table 5.1: Overview about safety (p_s) and liveness (p_l) properties for legitimate queries and their forged counterpart expressed by p'_s and p'_l. $< \epsilon$ is used to indicate very small values.

By construction of a formal model in terms of a probabilistic system, safety property is formally verified using PRISM. Especially the safety property can be used to argue about the quality induced when using a specific parameter setting. Table 5.1 summarizes the protocol properties for the AQF and sAQF protocol. It is important to notice, that the safety- and liveness property for the AQF algorithm are below 1 meaning that there can be legitimate queries rejected by a node and hence these will not reach the whole network. In addition the properties for forged queries are summarized which fulfill the safety property p'_s which equals the respective forwarding probability and forged queries reach in both cases only a small part of the network indicated by $p'_l < \epsilon$.

As in the present situation there is a big difference on the induced security mechanism that depends on the authenticator bits. Hence there is a fundamental difference whether one bit, two bits or more authenticator bits are checked by a node. Since an adversary can guess one bit with a probability of one half, the probability to correctly guess two bits already decreases to one-fourth. This quality assurance is provided by the figure of common keys ($CommonKeys$) between the authenticator and a specific node.

When comparing sAQF with the AQF protocol [BFH+06] from Section 4.3 on page 56, the main difference lies in the authenticator. In the sAQF protocol all keys from the keypool (ℓ bits) are used, and hereby all nodes can check the legitimacy of the authenticator using always k bits from their key ring. In contrary, for the AQF algorithm a randomly chosen subset of the keypool is used to construct the authenticator. This disadvantage of this is as stated before that liveness and safety properties are not fully satisfied.

In the present work the application of a model based verification approach did especially reveal the advantage of rigorous mathematical proofs in the field of probabilistic systems. They have proven to be durable in situations were the occurrence of errors occurs with some probability in contrary to their fragile simulation based counter parts. Much of this success is due to the sound and mathematical foundation.

Contribution Using rigorous techniques as offered by formal methods helped in finding potential errors in the performance and correctness analysis of the query dissemination protocol. Within the present work there were errors hidden in the simulation script that caused erroneous data. This fault was highlighted by the development and evaluation of a formal model. Essentially, this course of action shows that a higher degree of confidence about a model or formula can be reached when using several distinct and independent investigation methods. Hence the advantage of each approach can be beneficially used and incorporated in the final results. As in the present situation, it is very likely that the error in the formula would be hard to discover using simulation alone due to its variation. Since theses results have a strong impact on the qualitative statements about the developed protocol, they are deceptive and lead to wrong conclusions.

Furthermore this work analyzed the derived random-set theory formula and discovered the neglected treatment of a dependent variable. In addition the neglected dependency constraint of the random-set formula that led to a minor but still present error in the results is shown by the application of formal methods. Theses faulty assumption leads to results that would not have been discovered by the pure use of simulation techniques. Even with a correct simulation were the preliminary errors are fixed the obtained results have such a variation that makes it hard to compare against mathematically computed figures. In consequence results are misleading due to their very deviation from the mean.

Using a defective statement like the p_f formula in the present scenario results in delusive conclusions about the efficiency of the developed algorithm.

Since the random set formula cannot be fixed, we investigated and derived a new formula to compute the correct probability which is based on the evolution of the hyper-geometric distribution. Its properness for a selected range of input variables is proven albeit we do not give a total correctness proof covering all possible input parameters.

In the present chapter we gave evidence using a probabilistic model manually designed, that the derived hyper-geometric formula is correct in a sense, that it computes the probability for forwarding a forged packet in an accurate way. In addition we formally verified a safety property of the AQF algorithm from the previous chapter by construction of a suitable model.

In general for manually created artifacts it is difficult to state guarantees since whenever human intervention is involved, there is the potential possibility of errors. In addition the modeling process is time consuming and an annoying matter. With regard to this, an automated model generation has many advantages, but in turn the models become enormously complex since the abstraction level is reduced or even completely removed.

In the following chapter we investigate a generation mechanism which translates models into software behavior models that can be verified using conventional software verification tools. We exemplify this idea using a concast protocol for parameterized and secure data aggregation.

CHAPTER 6

A Verifiable and Secure Concast Protocol

6.1 Introduction

The design of secure and dependable protocols is nowadays a challenge to software engineers, and not only restricted to emerging technologies in the field of safety critical applications. As a fact, embedded systems moved from niche products to ubiquitously present systems which fulfill the demand of highly distributed and real-time applications. Since theses systems gain an increasing relevance in daily life, there is a strong urge to ensure that these every day applications on which we rely become more dependable and secure.

The application of formal methods to embedded systems could be the key to overcome this dilemma, but it is tempting to assume, that verification in the area of small gadgets is nowadays nothing but a push-button process. Although embedded devices carry only several hundred kilobytes of memory — which is in comparison with a modern operating system on personal computers relatively small — proving formal correctness is far from being an easy task. However, considerable progress has been obtained in the field of hardware verification, and software correctness checking and appropriate tools exist.

In fact, embedded software systems are very difficult to verify due to their concurrent nature. Very often they employ a behavior which uses heavy interaction of software and hardware through common channels, for example like I/O register. For this reason the software does not need to be considered on its own, but in combination with a well suited hardware model underneath. Another challenge in the context of embedded devices is that they are deployed to autonomously interact with their physical environment by obtaining external stimuli to the hardware and respond with an adequate action on that. This continuous monitoring and task execution is difficult, and since formal models need to be finite in size, an abstraction is required. In addition a sophisticated modeling has to fulfill an appropriate abstraction from essential features. Furthermore, the proper definition of specifications is essential to tackle the verification task.

In this context, wireless sensor networks can be understood as embedded devices which employ nearly all characteristics from embedded systems in a single product. In addition they run complex algorithms for encryption, distributed data management (i.e., TinyDB [MFHH05]), and wireless communication making them a challenging but perfect candidate for the application of formal methods. In this sense we consider for a model based verification the simultaneous access to the wireless medium, and an appropriate modeling of implemented algorithms to be essential challenges in providing correctness proofs of the software implementation and the underlying hardware. Yet a comprehensive model requires refinement from its real world counterpart, a contradicting matter since the level of abstraction inversely influences the complexity of the model under consideration. So, the closer the real-world system is mapped onto the model, the more realistic is may appear but in turn the formal analysis is gaining complexity, making thorough checking impossible.

By now different approaches have been established when considering the formal correctness analysis of distributed software systems. On the one hand these instances can be checked with the help of theorem provers also used in the traditional software verification process. These tools can nicely be used on infinite state systems and in particular by their abstract representation of discrete values, they are very powerful tools. In addition to this there is either an automated translation into an intermediate language or the theorem checkers can directly be applied to the source code. Very often the user has only to provide the specification and the tools can construct a proof almost automatically. Either way, this group of verification tools has a significant drawback, that is gaining importance in modern computer systems: They cannot handle threads, and consequently the verification of the concurrent behavior of modern multi-tasking systems is impossible.

Here another course of action comes into play, the so called model based verification approach which subsumes model checking. Model checking is based on a model that the user has to provide, hereby influencing the degree of abstraction and the level of detail but in turn the manual model building may also introduce errors. And in particular when the specification is available previous to the model design and the features under investigation are well known, the later model can abstract from insignificant matters and solely contain the interesting mechanism. The theory of model checking is eminently useful for the proof of distributed systems where concurrent and independent events can simultaneously occur.

Yet, the application of model checking as well as other model based approaches did not make it into development cycles in industry since it exhibits some drawbacks. First of all, they are prone to errors since the model needs to be manually inferred from manuals and descriptions and in-detail knowledge and experience is often required. The development of models is costly since the artifacts are only required for the verification and cannot be used further during the development process of the product. In addition it is indisputable that especially during the design phase blue-prints of the product rapidly change and hence there is additional work required to keep the verification model and the implementation in sync. The most important fact is however the danger to abstract from fault-prone details due to missing constructs in the modeling language or human misconduct and in consequence these errors will not be revealed. Due to these reasons, a method is required which automatically generates models for the verification process to eliminate all of the human introduced errors and reduce the risk of mistakes to a minimum.

In this chapter a concast protocol [CGSW99] for secure and parameterized aggregation called ESAWN [BWZ08] is considered. By concast we mean essentially the inverse of a multicast that uses a best-effort service to accomplish the delivery of packets. So in contrary to a multicast paradigm, in concast a receiver merges packets obtained from multiple senders at confluence points in the network. Hereby a notable amount of energy can be saved since instead of sending multiple packets, the confluence — the aggregation of individual packets according to an aggregation function — is sent.

For the verification task a combination of software verification and model based verification is opted to benefit from both techniques. To guarantee safety and liveness properties that originate from the distributed character of the network, a SPIN model is developed and selected specifications are verified. Since the SPIN model checker has means to express the concurrent nature of wireless sensor networks it is promising to find potential errors.

The second analysis will complement the distributed verification and account for the thorough checking of generated software sources that fully comprises the behavior of the sensor node. Afterwards, the formal verification is accomplished using software bounded model checking (see Section 2.3.4), a technique that is not related to model checking in the classical sense although the terminology suggest so. Instead, the software behavior is encoded into a SAT instance which is checkable for satisfiability using SAT solvers.

Related Work In the work of [KMG08] a formalism is described which automatically derives a high-level system logic from low-level TinyOS programs. The technique of symbolic execution is used and adopted to handle the event-driven nature of the TinyOS framework. The formalism is finally employed in a tool called *FSMGen*. In particular generic components are provided that approximate the behavior of sensor network components. The resulting finite state machine representation of components is obtained by predicate abstraction.

In the work [Han07] a tool called *Slede* is proposed for error and bug detection in security protocols. This tool automatically extracts a model from a provided nesC implementation. In addition the user defines a topology and the verification goals. Out of the protocol specification, an intrusion model is automatically generated. All of these models are combined using a protocol model generator into a Promela model. By the use of SPIN the final model description is verified. An obvious drawback of the used approach is the implemented intrusion model that only allows the Dolev-Yao threat model. In addition it turns out that only software written in TinyOS 1.0 can be used as input for the verification tool.

The *FeaVer* verification system [HS00] offers means to mechanically extract a verification model from implementations in C, guided by user-defined lookup. Hereby essential parts are either replaced using a non-deterministic function, irrelevant parts are suppressed, and not considered statements are stripped from the model. The verification is performed using a standard logic model checker. The tool is exercised using a call processing software for an access server called PathStar.

KLEE [CDE08] is a symbolic execution tool for the analysis of embedded C code, generating high coverage tests. It is capable of interaction with an environment and its constraint solving optimization can deeply check applications using a space efficient representation of checked paths. By applying efficient search heuristics, paths are chosen to obtain high code coverage. Still the code coverage is not 100% in contrast to classical model checking approaches and consequently not all traces violating a property are found.

Overview This chapter is structured in the following way. In Section 6.2 we give a short introduction to the investigated aggregation protocol, its parameters and a simple application scenario. Thereafter in Section 6.3 a distributed scenario is modeled with SPIN that includes a network mode, an adopted adversary from Section 2.4.2, and the models of different sensor nodes. A correctness proof is provided which especially accounts for the distributed nature of the protocol.

The model in Section 6.4 covers the behavior of sensor nodes which is derived from software sources. Furthermore, the specification of software sources and final verification using software bounded model checking is presented. Results are collected in Section 6.5. Afterwards the chapter is concluded in the last Section 6.6 summarizing and comparing the results of both approaches and the feasibility of software verification in embedded systems and especially in wireless sensor networks.

6.2 Secure Aggregation in Wireless Sensor Networks

We consider a concast protocol [CGSW99] that is extended by a probabilistic relaxation of authenticity called *ESAWN* [BWZ08] (*Extended Secure Aggregation for Wireless sensor Networks*). It is different from previous work since it handles the transport and aggregation of messages with guaranteed end-to-end authenticity in the presence of multiple malicious nodes. By malicious nodes we consider devices that are in the network but under the control of an adverse entity (see Section 2.4.2). Since authenticity of the aggregated data and energy preserving mechanisms are in the focus of this protocol, it offers these parameters to influence the behavior. This means that in case the user is willing to relax the credibility constraint of the data by allowing data to be authentic according only to some probability p (i.e., 50%) instead of 100%, a notable amount of energy can be saved. So the tradeoff that is involved hereby is that sacrificing energy at the cost of authenticity of the collected data and vice versa.

Essentially this energy preservation is accomplished by accumulating software packets as done in concast protocols and in addition with a probability $0 \leq p \leq 1$ witnessing nodes on the route to the sink are used to attest the authenticity of the sent data packet. Since the use of witnesses increases the energy draw, the choice of the parameter p influences the overall energy savings. The second parameter of the protocol specifies the number of witnesses to be used which must always be more than the number of malicious nodes since a forged aggregate can otherwise not be detected.

Instead of sending a data packet, data values are pre-processed by the use of an aggregation function f_{agg} that can be for example the maximum function or anything else, depending on the requirements. When considering Figure 6.1 a linear aggregation tree is shown. Here the $leaf$ node (n_4) starts to send the data value A to node n_3. Since node n_3 could also be compromised, data is also sent to w consecutive nodes on the aggregation path. For two witnesses ($w = 2$) that is node n_2 and n_1. Each node n_i which receives a packet computes the aggregate $f_{agg} = f(n_i, n_{n-1})$.

In this way energy is preserved since data is already aggregated in the network and hence fewer radio transmissions are required in comparison to standard multi-hop communication. Obviously, the security related problem that exists is that adverse nodes

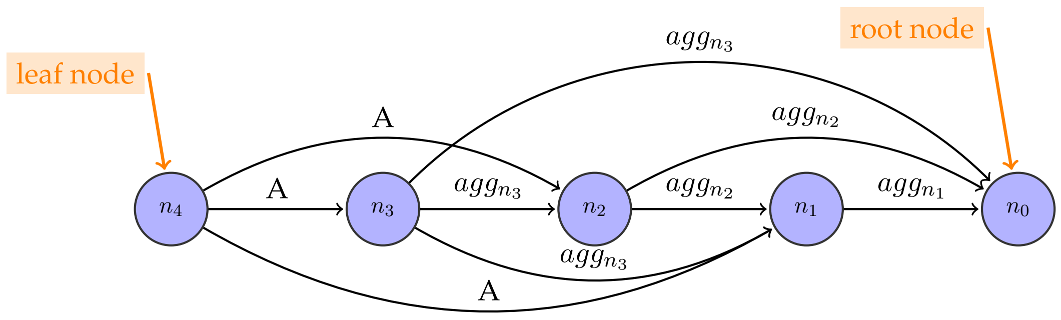


Figure 6.1: ESAWN scenario of an aggregation tree with parameters $w = 2$, the original data value A from node $leaf$. The data from node n_3 in aggregated with the value A in the packet agg_{n_3}.

may cheat be forwarding wrong computed aggregate. Such a fraudulent behavior can be discovered by the use of witnesses and hence the authenticity of the aggregated value is verifiable all the way down to the root.

6.3 A Correctness Proof using the Spin Model Checker

6.3.1 Network and Intruder

As opposing to the ESAWN model where a probabilistic relaxation of the authenticity is considered due to energy, we assume that in the SPIN model every node forwards the aggregate with probability 1 to witnessing nodes. This is motivated since probabilistic behavior is difficult to model within the SPIN model checker. Since we are not interested in quantitative properties to be checked — like what is the energy when using a probability of 5% — modeling probabilistic relaxation of the authenticity property is not considered here.

In the selected scenario nodes are lined up as Figure 6.1 depicts. In fact this is a special case in which the protocol is not working most efficiently since on every node the aggregate is built using only two values, not a very efficient application case of ESAWN. But due to modeling reasons, this topology essentially represents all of the desired properties, while being relatively easy to verify. In addition, it can be proven that the same properties hold here like in a more tree-like setting, and hence this topology is a good candidate for the SPIN model.

The use of ESAWN's aggregation function does not need special modeling in SPIN as well as the computation of parent nodes and the computation tree since in our setting every node n_i aggregates and forwards to n_{i-1} up to n_{i-w} nodes for w being the witness parameter from before.

6.3.2 Adversary Model

The abilities of the adverse entity are already described in Section 2.4.2. It can compromise some nodes in the network by reading out the nodes' memory and obtain their secret keys, reprogram nodes, inject its own code, and place nodes again undetected back into the network. In addition nodes are completely randomly chosen by the adversary and the only two nodes fixed are leaf and root node.

In this sense we expect the root and leaf node to be out of the adversary's reach, operating honestly and always compliant to the protocol. The reason for this is that if the root node would act maliciously there would be no meaningful verification of the authenticity possible since the user could not trust the base station as already stated in Section 2.4.2 on page 23. Furthermore, if the adversary has control over the root node, it has all means to take over control of the whole network by sending his own request which will possibly not be recognized by the user. This does also hold in a similar way for the leaf node. Here one could never check whether the external measured data is correct or forged by a physical manipulation of the hardware sensors. This could for example be done by simply using a lighter at the node's temperature sensor thus artificially raising the measured values and pretending wrong results.

In addition it is assumed, that if the adversary wants to compromise, he coordinates its nodes and starts the attack simultaneously by exposing a protocol non-compliant behavior, and thus amplifying the impact of his attack. In particular by the use of out-of-band communication means that make the knowledge of one node immediately known to other nodes, the severity of the attack can be increased.

6.3.3 Modeling the Network

In the Promela definition the real world ESAWN protocol is modeled by the initial definition of the network size by a parameter N. By variable k the number of compromised nodes is denoted that counterfeit packets from time to time but operate normal and inconspicuous most of the time. Parameter w indicates the witnessing nodes, i.e., a node has to send each packet to at least w parent nodes which verify the correctness of its aggregates.

For the network scenario four types of nodes are considered. The $leaf$ node is in charge of doing measurements and sending the initially collected sensor data to the network. Among inner nodes (nodes $n_1, \ldots, n_3$ in Figure 6.1) two different types of nodes are considered which are $InnerNotCorrupt$ and $InnerCorrupt$ with a behavior as suggested by their names. The node at the sink is the $root$ where the aggregated data is received and made available to the used.

Channels interconnecting Nodes

Before the ESAWN protocol actually runs, each node initializes the required message channels on the aggregation path. This means that a node obtains input channels from children nodes and allocates outgoing channels to parent nodes on the aggregation path. Channels are modeled as unbuffered 1-byte variables of type `chans`. The use of separate channels is legitimate and can be motivated by the pairwise symmetric encryption in the ESAWN protocol using *SKEY* (*Secure KEYing*, see [BCZ05, ZB06]). In consequence no other but the destination to whom a message is addressed is able to decrypt it and read its content. In the SPIN model these requirements are met by the use of channel assertions (see Section 2.3.1) that guarantee exclusive access of the process to whom the channel belongs.

```
    i = 0;
    do
    :: else ->
         /* send */
         ch = channel[myid - (i+1)].c[i];
         /* generate any data */
         if
              /* correct data */
              :: data = 0;
              /* incorrect data */
              :: data = 1; FakePacketsSend++
         fi;
         ch!data;
         i++;
    :: i>K || i>=myid -> break;
    od;
```

Figure 6.2: Promela code for a compromised node.

State Variables

The overall state of the model is expressed by global variables which are used to specify the LTL proof obligations. $CheatingDetected$ represents a discovered faked packet, only detected and issued by nodes $InnerNotCorrupt$ and $root$ since the adversary has no incentive to expose an attack.

By $AnnounceReceived$ the $root$ process declares the reception of a packet. If the received packet is as expected variable $ReceivedDataCorrect$ is set to $true$. The number of fake packets is counted by $FakePacketsSend$ and increased whenever an additional fake packet is sent.

Processes

In the initial configuration all four processes setup their channels with the designated child and parent node on the aggregation tree. The $leaf$ node afterwards sends a sample with value 0 that represents an initial sensor value. $InnerNotCorrupt$ nodes behave loyal with respect to the protocol and receive all the packets and compare all of them for equality. Essentially if $aggStore[i]! = aggStore[i-1]$ then variable $CheatingDetected$ is set to $true$. Otherwise the results is sent to subsequent w nodes.

$InnerCorrupt$ nodes behave different in a sense that they forge packets and instead of forwarding value 0 they forward 1. Forging occurs on a packet basis, meaning that for every packet corrupt nodes decide whether to forge or not. Since compromised nodes cooperate, they will not reveal an attack if they receive faked aggregates. Finally the root node checks for the consistency of data as the $InnerNotCorrupt$ nodes. In addition, if the last pair of results is validated and all packets are checked correctly, variable $ReceivedDataCorrect$ is set. Upon reception of the last expected packet, the root node issues an $AnnounceReceived$ and all processes terminate. The relevant Promela code is shown in Figure 6.2.

6.3.4 Security Related Properties

For checking the properties no assertions are used. In the following four properties of interest are specified using LTL.

For a proper function of the network it is important that no message is lost. So we expect that either the root receives data ($AnnounceReceived$) or one or more nodes are cheating which is detected by at least one node ($CheatingDetected$). The use of "one node" is sufficient since it will trigger already the alarm. This property is specified by:

$$\Diamond(AnnounceReceived \vee CheatingDetected) \tag{6.1}$$

In the following property we denote that either correct data is transmitted in the presence of corrupt nodes or a forged aggregate is detected. This means that if the root node ever receives a data packet ($AnnounceReceived$) it is either identical with the one sent by the leaf node and no faked aggregates sent ($ReceivedDataCorrect$) or at least one node detected a corruption ($CheatingDetected$):

$$\Box AnnounceReceived \rightarrow (ReceivedDataCorrect \vee CheatingDetected) \tag{6.2}$$

Essential for the ESAWN protocol is that whenever there is a corrupt packet sent ($FakePacketsSend > 0$) this will eventually be reported:

$$\Box(FakePacketsSend > 0 \rightarrow \Diamond CheatingDetected) \tag{6.3}$$

In addition, the following property states that whenever the data packet received by the root node is correct, there has been no corrupt packet send ($FakePacketsSend > 0$) although malicious nodes might be present:

$$\Box(ReceivedDataCorrect \rightarrow !(FakePacketsSend > 0)) \tag{6.4}$$

Before we present the results from the distributed model in SPIN, the second approach is discussed which automatically generates the behavior of a sensor node in terms of software code. Afterwards the results of both solutions are presented and compared.

6.4 Software Model Checking of Embedded Software

In the previous section the topic of a formally verified protocol is based on a manually created model. Since especially the process of abstraction is crucial and error prone we propose in the following a way of automating the code generation from TinyOS (see Section 2.4.3) based implementations. We identify several ways of deriving software sources and discuss their advantage in detail. The related specifications are derived from the protocol requirements. Together with the most appropriate software model, the specifications are investigated and verified using bounded model checking.

6.4.1 Generation of Software Sources

We essentially point out three approaches to obtain a software model from the software implementation of the ESAWN protocol using the built-in nesC compiler. In fact, the derived software model is an intermediate stage in the software deployment process. This means that software in TinyOS is built up out of components which are linked using the nesC compiler with hardware specific modules before they are compiled and deployed on real hardware. So before the final software for a platform is compiled, the corresponding sources of the implementation are built. Note at this point that each hardware platform requires individual building and linking of the hardware specific modules that handle i.e., access to hardware specific registers, interfaces to the internal bus architecture, and the incorporation of specific chipset.

As it turns out each of this proposed software generation steps has pros and cons that have to be traded against each other. The considered model derivations are essentially the *MicaZ model* using the MicaZ platform as hardware model, the *TOSSIM sources* which produce software sources and libraries in C and Python for the network emulator TOSSIM, and finally the *NULL platform*. What is unique about the NULL platform is that all hardware dependent routines like access to hardware dependent I/O registers as well as the transceiver are abstracted.

It is worth to note that one could always manipulate the generated sources by hand to obtain a simpler model of the protocol. But this is contradicting with the earlier mentioned thought to automate the software derivation and generate models which do not require any human interference and which can directly be fed into a formal verification tool.

MicaZ Sources

The first presented option is to obtain the software behavior model form the MicaZ hardware platform (`make micaz`). This generated software can in this form be very well compiled and deployed on actual MicaZ nodes. The problem with the sources is just that many TinyOS functions are still present, for example constructs that calibrate the hardware clock, functionality to access the ADC (analog digital converter), UART etc. In addition the implementation makes frequent use of registers which are normally provided by the hardware, but missing and undefined in the obtained software code. Finally the concurrent behavior of the software model is lost due to the missing simultaneous and independent execution of software and hardware.

Considering the verification of this generated implementation it turns out that a hardware model would be additionally needed to fully specify and verify the software. At the same time, it is very likely to assume that due to complexity considerations, a combination of software and hardware showing a concurrent behavior would be much to complex for a deeper analysis.

TOSSIM Sources

The second investigated way is to generate software sources used for emulation in the TOSSIM simulator. In addition to the software sources, an executable simulation framework is hereby added to the source code. The advantage of this implementation is that the hardware is already abstracted and consequently there is no use to access registers for controlling the program. In addition ADC models, and a proper abstraction of the communication is provided. Since in the TOSSIM emulator node's internal clocks are directly controlled using the simulator instead of the node hardware clock, the generated software promises a good start for the verification.

The downside is that parts from the software implementation are written in C, others are written in Python to connect the simulation engine. In addition there are many library references that also hamper the verification process and add complexity to the software under verification. Of course, there would also be an optional start for a manual modification of the sources but due to the afore mentioned properties of manually written or modified software this not the best way to proceed.

NULL Platform

The last investigated way is the actually used one. It suggests the use of the NULL platform which is shipped with the TinyOS package and generates a hardware independent software model. In particular the platform can be understood as a skeletal structure containing only the functionality of the protocol plus some overhead like scheduling functions for jobs, the jobs queue and so on. But all hardware specific functions are abstracted and generated with an empty function body (e.g., UART, LEDs, ...). This has the major advantage that no platform must be considered when specifying the properties.

Even here some disadvantages exist. Since the platform is hardware independent, the functionality for the transceiver is lost and removed from the sources. In the ESAWN protocol under investigation nodes depend excessively on the use of packet sending and receiving, the empty function bodies need to be manually completed with the correct packet handling information. This manual manipulation is error prone since errors can be introduced hereby.

Additional open questions are the treatment of concurrent behavior which stems from hardware-software interaction. This means for example that no external interrupts can be issued by the hardware to signal events and preempt tasks. In turn, since no preemption is possible also the atomic statements can safely be ignored and removed which further reduces the complexity of the protocol.

Summarizing the three proposed approaches, the NULL platform deliveries the best start for a further analysis using a verification tools. By the use of the NULL platform we were able to reduce the complexity of the implementation from originally 21 000 lines of code as in the case of the MicaZ platform to 4 500 lines. In the following, specifications are defined that need to hold for a sound protocol execution.

6.4.2 Requirement & Specification

For the specification purpose an autostart function is added to the sources that starts the protocol with well defined parameters right after the initialization. Whether the values set by the autostart routine are correctly set can be checked using assertions at the end of the execution in the main function. Furthermore the protocol has a built-in alarm function which indicates a wrong behavior of the protocol or reveals a potential attack. By adding assert statements to this function erroneous packets can be discovered and a protocol unspecific behavior is revealed.

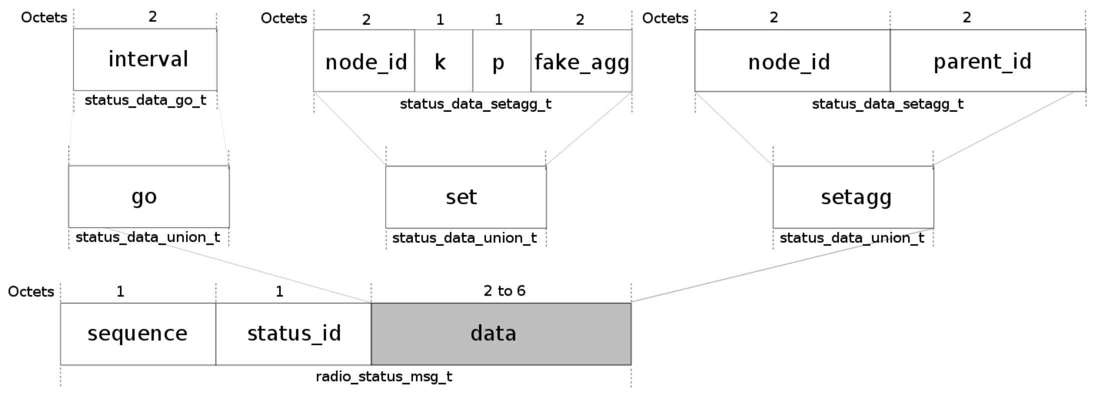


Figure 6.3: Structure of verification relevant `status` packets.

In addition to the sources used for the verification, we maintained an optional version of the node behavior with debugging statements. This was in particular useful for the analysis and the specification of properties because instead of verifying properties using the model, the sources could be compiled to obtain a simulation run of the ESAWN protocol with the current parameter setting.

What needs to be specified in the protocol is the correct treatment of the possible packets which are used by the ESAWN protocol. Essentially two kinds of packets exist. On the one hand `status` packets are in charge to initially transmit information of the network topology, the protocol parameters, and the parent-children relation which is required to correctly compute and verify the aggregates. The other type of packets are *ESAWN packets* used during protocol runs to transfer the aggregates through the network.

Status Packets

In the specification we concentrate for the moment on packets that deliver the major functionality of the protocol and ignore for example packets that broadcast alarm events, etc. Since the investigation of the implemented RC5 cipher is complex and probably worth another verification task, we consider for the moment unencrypted packets. Furthermore we omit logging messages constantly sent to the base station to describe the protocol state.

The first specification covers packets of type `status_set` which are sent initially by the base station to make protocol parameters known to the network. In detail a packet of this type contains the number of nodes in the network (`num_nodes`), the ESAWN parameter which determines the number of witnesses (`w`), the probabilistic relaxation parameter `p`, and a variable `fake_agg` to indicate whether the receiver should fake aggregates. The following property states that a node processes this type of packet correctly:

$$\texttt{status\_set}(num\_nodes, k, p, fake\_agg) \text{ is processed correctly} \tag{6.5}$$

The second type of specification considers packets sent to make the aggregation tree public. For these reasons each node obtains information about nodes to which it has a parent-child relation. This aggregation information is sent out using `status_setagg` packets. They are sent to every node in the network and contain fields *`node_id`* and *parent_id*. It is a critical point to correctly broadcast this information, and hence the following specification must be valid for a fixed node_id and all possible parent nodes on the aggregation tree:

$$\texttt{status\_setagg}(node\_id, parent\_id) \text{ is handled correctly} \tag{6.6}$$

Another important aspect is the information sent through packets of type `status_go` which initiate a protocol run by causing the leaf nodes of the topology to send data values to the sink. The contained value *interval* denotes the interval of two consecutive data values which are measured and sent. In case the interval is 0, the protocol in only executed once.

$$\text{there is an action upon receiving } \texttt{status\_go}(interval) \tag{6.7}$$

Of course other message like `status_reset` causing a hard reset of the node, or `status_alarm` can be specified, but we omit these trivial specifications here and keep it for future research.

ESAWN Packets

The ESAWN packets are used during protocol runs to exchange aggregates between nodes (see Figure 6.4). Since nodes relay packets to children nodes down the aggregation tree using their direct neighbor, this information is in the original version of the protocol encrypted using symmetric keys (cf. SKEY). For the correctness of the aggregation handling and forwarding of the protocol, it is sufficient to check, whether a well defined packet is treated correctly. In fact the protocol conform handling of a packet implies the correctness of all involved functions. The functions involved in the computation of an aggregate are the computation of the relay count which indicates to a node to how many children it should relay the aggregate. Also functionality for the computation of the aggregate is relevant in this case. As aggregation function f_{agg} the simple sum is used ($f_{agg}(a, b) = a + b$).

For the verification of ESAWN packets the topology from Figure 6.1 is assumed. As in the previous case, an autostart function initialized the network parameters using a `status` packet with parameters $w = 2$, $p = 1$, and num_nodes=5. In the specification definition we consider node n_2 in the aggregation tree which is expecting w aggregates from its parent nodes and in consequence sends w packets to children nodes. We further assume that means for external measurement are abstracted therefore each node sends as data value its node ID. Since it is unique it does not introduce any overhead to the model.

We consider the following packet which is sent to node n_2 for authentication and aggregation. It contains a sequence number, the originator of the packet in the field `from`, and the value in plain text in the field `value`. Since no encryption is used, the field `tan` can be safely ignored.

$$\text{packet } ESAWN[from = 3|value = 7|from = 4|value = 4] \text{ is handled correctly} \tag{6.8}$$

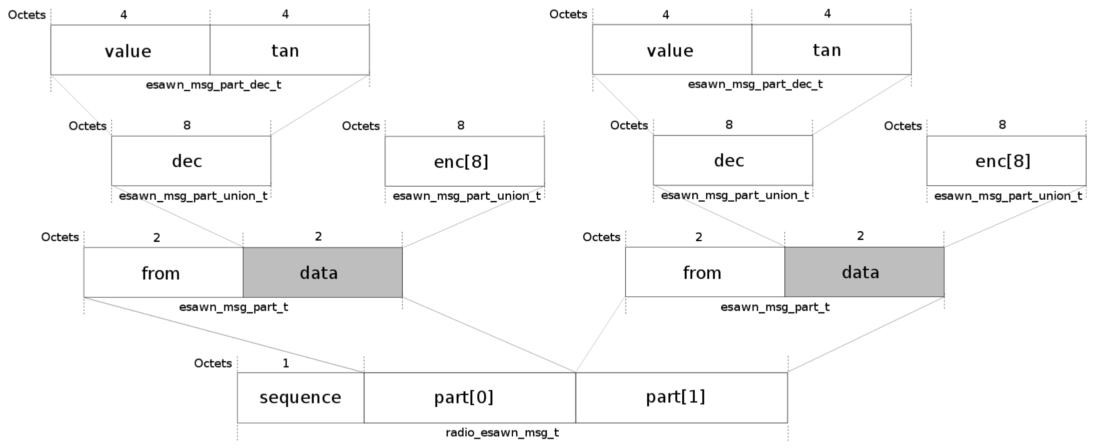


Figure 6.4: `ESAWN` packet definition

Especially if the protocol should properly function, no ESAWN alarm should occur which is monitored by assert statements in the alarm function.

Further, a node which has to verify an aggregate agg receives data packets not only from its immediate ancestor, but also from source nodes of its ancestor. If it receives multiple aggregates agg_{n_i} to agg_{n_j} is has to compute its own aggregate and compare it to the received packets, such that for any valid aggregate $agg = agg_{n_i} = \ldots = agg_{n_j}$ has to hold. This means that the aggregates a node receives from its ancestor have to be correct:

$$\forall i \leq w : (agg = agg_{n_i})?true : Alarm(8) \tag{6.9}$$

This property states that all w aggregates must be considered by a node, and if the equal checking fails, a *raise alarm* message (`Alarm(8)`) has to be issued and sent to all $w + 1$ ancestors in the aggregation tree. In this case it is either possible that the direct ancestor of the node forged the aggregate, or a non-direct ancestor — possibly any node up to k steps away from the verifying node — faked the aggregate. The correct handling of an alarm message can be checked using the outgoing packet queue of the node. In fact for a node i there must be $w + 1$ alarm messages send to nodes $n_{i-1} \ldots n_{i-w-1}$ in the outgoing queue.

$$if(\text{alarm at node i}) : \text{ alarm messages are sent to } n_{i-1} \ldots n_{i-w-1} \text{ nodes} \tag{6.10}$$

6.4.3 Treatment of Model Insufficiencies

Up to this point the software still contains construct which are not longer used. In this sense still functions for initializing the scheduler queue and the assignment of the empty task element are present. This functionality introduces additional complexity. Further, there are still routines for initializing the hardware platform present which are called before the actual program is started. In addition the software is initialized, meaning that serial interface for sending and receiving is initialized, the random seed is computed, and internal timer are calibrated. All of the above mentioned can be safely ignored for the verification task, and therefore we simply take a short cut and remove all of this functionality since it has no impact on the protocol behavior under investigation.

Another issue considers the nature of embedded systems. In principle the ESAWN software is designed for continuous operation. This means that the task scheduler is permanently executing jobs in the task queue. If there are no jobs in the queue, the node is going to sleep mode for a defined time span. Afterwards it wakes up to check the scheduling queue again. If a job is found in the queue, it is executed.

This continuous queue checking, sleeping, and execution of tasks makes a direct correctness check impossible due to memory restrictions. In particular the verification task is only feasible for a finite set of states and terminating programs. However, the source code does usually not terminate due to its controller like nature: once the node is switched on, it does permanently interact with its environment, sensing data, generating and processing new packets. In contrast to this finding any setting with a bound number of processed packets can be covered using a terminating abstraction. It can even be formally proven, that the implementation is still correct since individual messages are processed and received protocol conform.

For these reasons a modification is inserted into the software which executed a fixed number of jobs, hereby introducing an upper bound for the complexity. After the bound is reached, the node stops any further execution and terminates.

Another shortcoming of the model concerns the analysis of protocols running on different nodes, physically distinct by their location. The derived software does not account for a distributed setting where messages are interchanged. To imitate such a setting with more than one node and the interchange of information, we propose the following solution. By interchanging the node's ID at runtime we can imitate the distributed processing of packets.

For example consider a node with ID i which has a packet to send to node i'. In this sense, a packet is built which holds the destination address i' and is enqueued into the task queue. In the real TinyOS implementation, packets from the task queue are frequently probed and sent using a dedicated sending tasks which copies the packet from the packet queue to the transceiver queue. From there the packet is transmitted using the hardware chip.

The proposed modification starts with the packet in the sending queue, and instead of calling the send routine, the node ID is changed to i'. In case the send task is called, the node detects that the packet reached its destination, namely node i'. In effect, instead of sending the packet, it is processed by the same node which changed its ID from i to i'. Using this revision of the source model, a distributed network behavior can be imitated.

6.4.4 Software Bounded Model Checking using CBMC

So far we provided a specification of an essential set of properties that must hold to argue about a proper functionality of this limited set. Since the sources still use heavily pointer arithmetics, a verification tool is needed which is capable of nearly full ANSI-C source code. Such a tool is for example CBMC [CKL04], a software bounded model checking (SBMC) implementation for C programs. Before CBMC is described in detail, we will first review the technique SBMC. SBMC computes a solution to the following problem:

Definition 5 (The SBMC Problem) *For a program P, a bound k and a property f, does there exist a path p of at most length k that violates the property f?*

A *program state* can be characterized by the content of heap, stack, all registers and a program counter. A *path* is a sequence of program states where a transition between states is triggered by a C statement. One interpretation of the *length* of a path in a program is the number of statements in a program. *Properties* declare some error states, or invalid sequences of program states, that shall never occur in any execution.

Usually three possible results for the SBMC problem are implemented [CKL04]:

- The property f holds for all paths.
- The property f does not hold for at least one path p'.
- The bound is too small for at least one path p'.

In the two later cases a witness path p' is computed. This witness is a counter-example having the form of a concrete program execution. For a program that contains a finite set of finite paths, k can always be set large enough such that a sound and complete verification of the property f can be achieved. If the bound is too small it can be iteratively increased. Finite state programs that terminate for every input are relatively common in the domain of embedded systems.

CBMC (see 2.3.4 on page 20) implements SBMC for C programs. Properties have to be specified by `assert(f)` statements. The semantics of such a statement is that whenever a program execution reaches the statement, the condition `f` must evaluate to `true`. The semantic of the *bound* on the length of paths in CBMC is the following:

Definition 6 (CBMC Bound) *For* CBMC *the positive integer* bound *denotes the maximum number of loop body executions and the maximum recursive depth.*

The *recursive depth* of a path is the number of equal return addresses stored in the stack. For a given program, the bound limits the number of statements on any path. Note that in CBMC the bound can be set individually for each C loop occurring in the program.

The commonly used decision procedure in CBMC to decide the SBMC problem is a DPLL-style propositional satisfiability (SAT) solver. CBMC internally uses the SAT solver Minisat2 [ES03]. If the SAT problem is satisfiable, CBMC generates a concrete counter-example from the satisfying assignment produced by the solver. If the SAT problem is not satisfiable then the property holds for all program executions and the program always terminates.

6.5 Results

This section covers results from both the distributed analysis with SPIN using a manual design with concurrent processes, and the formal analysis of generated software sources from TinyOS using software bounded model checking with CBMC.

property	states	transitions	memory [MB]	result
6.1	3 002 790	3 728 272	49.864	valid
6.2	1 352 775	1 720 044	50.450	valid
6.3	47 505	63 387	3.868	not valid
6.4	19 015	27 653	3.477	not valid

Table 6.1: Verification results for parameters $n = 5$, $k = 2$, $w = 3$

6.5.1 Concurrent and Distributed Analysis

The LTL properties 6.1 to 6.4 are checked using the model checker SPIN. In SPIN the memory of 512MB is guaranteed and partial order reduction is switched off. In addition an estimated state space size of $500 \cdot 10^3$ is specified with maximum search depth of $10\,000$ steps. The setting is fixed for a network of five nodes ($n = 5$), and two randomly placed compromised nodes ($k = 2$). This means in turn that 3 nodes behave protocol conform. Note in this context that the SPIN model's internal parameter `w` which accounts for the witnessing nodes has to be set appropriately and exceed k by one ($w = 3$), since otherwise the ESAWN protocol cannot work in the presence of malicious nodes and the verification of properties would fail.

The verification results are captured in Table 6.1. In fact it is only surprising at first glance that properties 3 and 4 are violated. This is mainly caused by the cooperation of compromised nodes. Since an adversary wants to remain undetected, a compromised node will not trigger an alarm upon reception of a fraudulent packet.

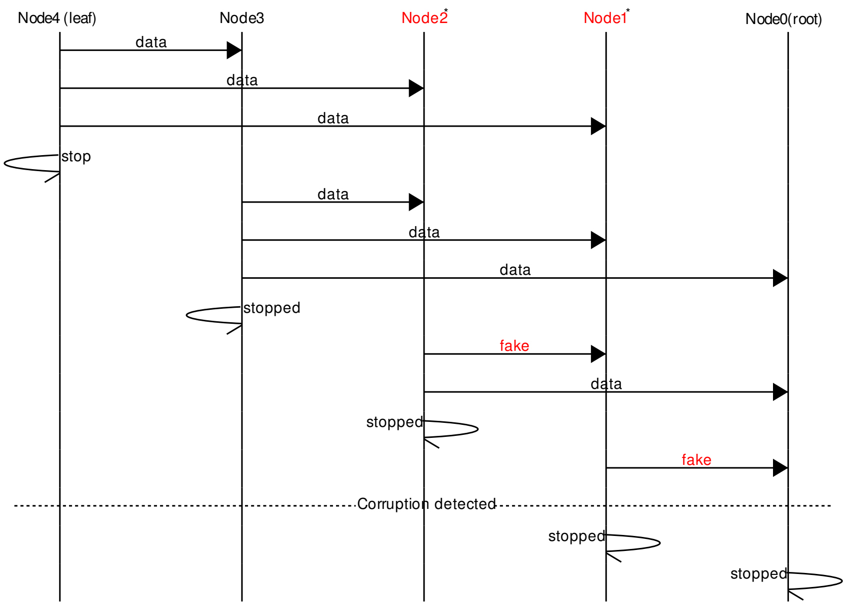
This situation is better illustrated by the message sequence charts (MCSs) in Figure 6.5 that display one possible trace of execution until the processes terminate. Variables `CorruptedNodeDetected` or `BetrayedPaket` are globally used and indicated using dashed lines. In detail, Figure 6.5a shows a protocol run where node 2 and node 1 are compromised. They both fake an aggregate and finally the corruption of the packet sent by node 1 is detected by the root.

A scenario where properties 6.3 and 6.4 are violated is depicted in Figure 6.5b. Here a fake aggregate is sent by node 2. Since we assumed that the adversary wants to remain undetected, node 3 will not trigger an alarm since corrupt nodes cooperate. In consequence, the root received two valid aggregates although an aggregate was forged in between.

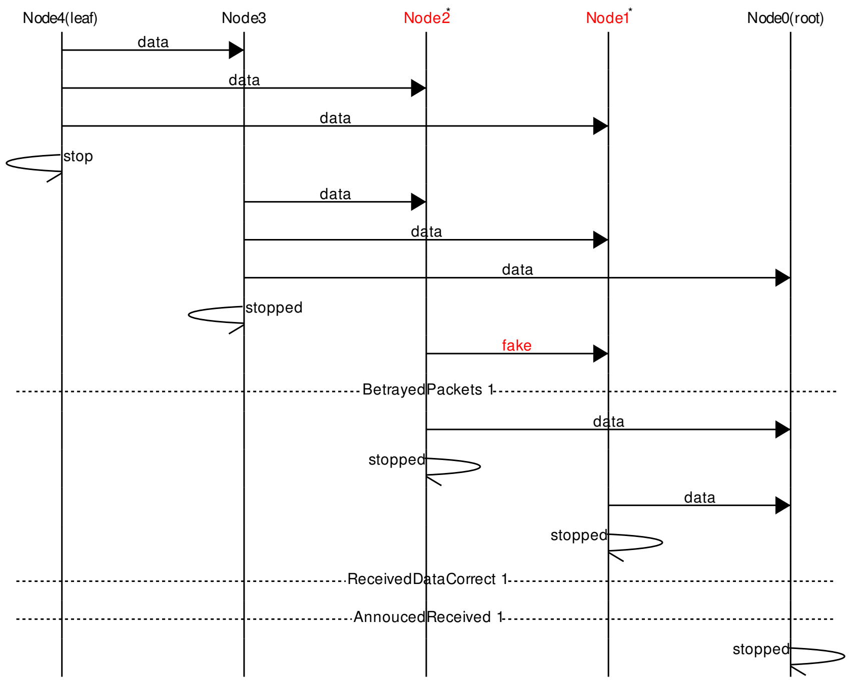
Note that this property does not state that we discovered a flaw in the ESAWN protocol. It rather pinpoints that a setting is possible where data is compromised and still the root node will receive correct data and not notice the fake aggregate.

6.5.2 Software Analysis

The generated software is equipped with the assertions which specify the above defined properties. Since packet types for `status` messages which are initially sent around are independent of ESAWN packets, they are treated independently. Before the respective properties are checked, it is important to find the maximum number of loop unwinding for each property. For this reason the unwindings are individually checked for each packet sent. Afterwards the remaining specifications are verified. The results for the considered `status` messages are displayed in Table 6.2. We use "?" to indicate that no results could



(a) MSC for parameters n=5, k=2, w=3



(b) scenario where property 3 and 4 fail.

Figure 6.5: Message Sequence Charts (MSCs) for ESAWN properties in SPIN.

`status_set`	check	successful	injected error	# claims
Property 6.5	unwinding	4	–	?
	assertions	yes	found	6
	bounds	yes	–	60
	pointer	no	–	181
`status_setagg`	check	successful	injected error	# claims
Property 6.6	unwinding	4	–	?
	assertions	no	–	4
	bounds	yes	–	60
	pointer	no	–	177
`status_go`	check	successful	injected error	# claims
Property 6.7	unwinding	4	–	?
	assertions	yes	–	4
	bounds	yes	–	59
	pointer	no	–	175

Table 6.2: Verification results for `status` packets for a valid loop unwinding of 4.

checked property	property	successful
correct packet treatment	6.8	?
correct aggregate computation	6.9	?
correct distribution of alarm messages	6.10	?

Table 6.3: Verification results for unwinding assertion using `ESAWN` packets.

be obtained due to memory restrictions, or due to a missing implementation in CBMC. The available memory was 32GB on the host system in use for the verification.

The results in Table 6.2 show that an unwinding depth of 4 is sufficient for each packet type. For Property 6.5 all but the pointer check are successful. This means that in turn, that when receiving a `status_set` packet, the node does correctly set the initial configuration. On the other side, the built-in pointer check failed for every packet type. We think that this is not a failure in the ESAWN protocol as these failures are caused by CBMC which does not find correct symbols during its pointer-analysis and in consequence there is either no target or no valid target assigned. In addition we inject erroneous assertions that are also found by the bounded model checking tool.

For broadcasting information of the aggregation tree using packet type `status_setagg`, in addition the assertion is violated, meaning that Property 6.6 does not hold. This seems to be a problem with the CBMC version at hand that is unable to handle e.g., arrays of structures.

The results regarding the sophisticated properties of `ESAWN` packets are captured in Table 6.3. Here the situation is disappointing since not a single property could be validated since CBMC did not reach a sufficient unwind. The highest unwinding tested was 11 and it took over 3 hours and about 30 GB of RAM to generate a counter example.

6.6 Conclusion

The presented results are twofold. We presented a formal analysis based on a manual created model and verified protocol related propertied using SPIN. It turns out that all critical properties hold which indicates that at least the specified behavior is valid in the model of the ESAWN protocol. The danger by using the term *valid* is, that the model fulfills the defined specifications, but the SPIN model is obtained by manual "design" which involves downsides. For example, there can be aspects of the ESAWN protocol which are modeled wrong since SPIN does not have appropriate formalisms to correctly reflect the intended behavior. In this sense, we cannot infer the protocol's correctness by verifying a manual created artefact. Hence a more realistic model is needed that is derived using almost no manual interference.

The second approach fulfills this criterion. Here a new software model is almost fully automatically derived from a description of sensor node in TinyOS, a development platform for sensor network software. Special about this model is, that is represents the behavior of a fully functional sensor node, defined in ANSI-C. Furthermore, the model has no level of abstraction since it represents the real node behavior.

For proving correctness of the ESAWN protocol, we defined important properties and added them to the software code by `assert(f)` statements. These properties are then checked using bounded software model checking. In the example at hand, we used CBMC (see Section 2.3.4) for this task which offers in addition to assertion checking also safety checks on the C code. Eventually we were able to prove some of the properties. In the example considering the assertion checking of Property 6.6 we also added errors to the code that were in turn found by verification.

In fact it turned out for the software model that the main problem is caused by the bounded model checker and its handling of the sending and receiving procedure. Since ESAWN heavily depends on message passing between nodes, CBMC was not able to handle the sophisticated property that covers the correct treatment of ESAWN packets. For the basic properties that cover the broadcast of the initial configuration, properties were successfully checked.

Although the proposed handling of complicated sending and receiving routines, that involve more than one node could be possible using another tool, we did not check on that. Instead we hope that a new release of CBMC will fix the missing formalisms and in particular enables the treatment of more sophisticated properties.

In the current case we were able to witness the complexity of automatically generated software from the TinyOS platform. Still, it requires some more abstraction to obtain suitable models which can be formally verified. At this point it is worth to state that besides the TinyOS platform also alternative node platforms exist, which run for example a virtual machine, similar to the concept of JAVA.

A candidate for software written in JAVA is the SunSPOT platform which is investigated by formal methods in the following chapter. Essentially, we declare in the following specifications for the SunSPOT's underlying networking library and investigate them using a theorem prover called KeY.

CHAPTER 7

Survey of the Sun SPOT's Networking Library

7.1 Introduction

Verification, specification, and excessive testing are elementary phases in the life cycle of any software product. Especially by giving a formal proof that the software fulfills the specifications, a desired degree of confidence is obtained. In the field of embedded devices (e.g., airbag controllers, control logics in avionics, etc.) which include safety critical software, the application of rigorous testing and verification methods is even essential.

Very often it turns out that the critical part of the verification is not the use of the verification tool. It is rather the combination of using the correct specification, the appropriate tool, and the expertise that finally lead to a successful software correctness proof. In particular the specification of "what should the system do?" and "how should it should react on failures?" is difficult to obtain since it requires a global view on the software product. Such a *global view* is very hard to maintain, due to the rising complexity and the fact that many software engineers from different disciplines contribute to the software with their expert knowledge. As a result it is difficult to infer specification from the software sources, even if they are well documented. What is likely to happen is that the use of an informal documentation written in English can be misunderstood, misinterpreted, and in effect a resulting erroneous specifications will very likely not reveal any error in the software code.

However, as the complexity of modern embedded devices increases, the development of sophisticated new tools and verification environments is required to keep up with the technological progress. The KeY tool (see Section 2.3.5) is such an environment that supports the formal development of object oriented software. It offers an integrated software design of object oriented software, its formal specification and - verification. KeY allows the verification of full JAVACard, a subset of the JAVA language. Furthermore, the KeY tool has proven to be an excellent candidate for the verification of object oriented software in previous case studies like [Ton07, BH05].

In the presented work [Sch08] parts of a commercial software library for embedded devices called SunSPOTs (Small Programmable Object Technology) from Sun Microsystems (see Section 2.4.3) is verified. The SunSPOTs are small mobile computers with a wireless network interface that run a JAVA virtual machine. What is special about the SunSPOT's software implementation is that most of it is written using JAVA and only minor parts that directly interlink the software to the hardware like I/O-library and native code are implemented using C.

For the reason that the SunSPOT's networking library is a complex artefact built of thousand lines of code (or even hundred thousands lines of code), the verification task is not an easy endeavor, and in fact to apply formal methods to the whole library would be a work for several person-years. For this reason only a part of the network stack — the so called LoWPAN (low-power wireless personal area networks) layer — is identified for the verification task. We believe and have strong indication to assume bugs hidden in this part of the library.

Related Work

The afore mentioned case study motivated the application of deductive verification methods on the SunSPOT's network library as conducted in the work of [Sch08]. To our best knowledge, there is no research that verifies object oriented software for embedded sensor node applications using formal methods. Within this context this work is a novel approach that verified a considerable portion of the SunSPOT's network library using KeY and proved correctness of the implementation with respect to a formal specification inferred from the available sources.

Other related work can be found in [Mos07] where a reference implementation of the JAVACard API is formally verified using a deductive verification approach. For this case study the KeY verification environment is also used, that allows a symbolic execution of the JAVA sources.

Overview

The work is organized as follows: In Section 7.2 a short overview about the SunSPOT's architecture is given, showing the relevant parts which are considered for the verification. In the following Section 7.3 we investigate a sending procedure from the networking library with related classes which is called whenever a packets needs to be send on the user level. An example of a verified method is given and related problems and potential short coming of this work are pointed out. The final conclusion in Section 7.5 wraps up the chapter at hand.

7.2 SunSPOT's Network Library

Before the actual verification task is discussed, the SunSPOTs networking architecture is explained which will help to classify the present work into the verification context and highlight its results. The implemented network layer model for the SunSPOTs consists of application layer, transport layer, network layer, data link, and the physical layer, just like the TCP/IP model [RFC89]. The MAC sublayer and the physical layer hereby comply to the wireless IEEE standard [Soc06]. In addition, the SunSPOTs have an intermediate layer — called the LoWPAN layer — that is in charge of providing special functionality not found

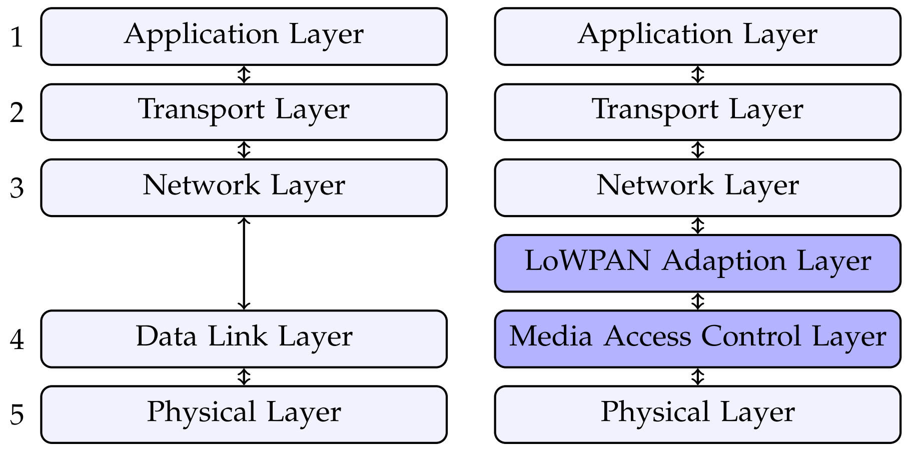


Figure 7.1: SunSPOT's network layer model.

in conventional wired architectures. It allows the processing of packets as defined in the low pan draft like link layer fragmentation and the option for mesh routing in combination with the routing manager. The layers of the TCP/IP model and the architecture of the SunSPOT's network library with the additional LoWPAN adaption layer are displayed in Figure 7.1.

7.2.1 The LoWPAN Adaption Layer

The LoWPAN adaption layer defines the payload of the IEEE 802.15.4 MAC protocol data unit (PDU) [MKHC07]. Its main functionality provides the encapsulation of frames from the above layer (IPv6 packets) and adds additional header information. Each LoWPAN encapsulation frame contains zero up to four header fields which provide information like *mesh routing* (mesh addressing), hop-by-hop options (broadcast/multicast), fragmentation, and the payload. An overview of the mesh addressing header, LoWPAN adaption layer, and the MAC layer is shown in Figure 7.2.

The mesh header is started by a one-bit followed by a zero-bit. The two flags that follow thereafter are O and F to indicate whether the originator address and the final destination address are specified as a 16-bit address or a 64-bit address. It is even possible to mix 16-bit and 64-bit addresses for origin and destination within a packet. The next four bits are reserved for the `HopsLeft` counter which is decremented before a node forwards a packet to its next hop. If the counter is decremented to 0, the packet is not sent any further. If a hop-distance greater than 14 is required, the field `HopsLeft` is set to `0xF` to indicate that the following 8-bit deep hops left field (`opt HopsLeft`) contains the hop counter. The final fields contain the 16-bit or 64-bit addresses of the originator and destination.

In the LoWPAN frame, the information which follows the *mesh header*, is stored in the broadcast header containing additional mesh routing functionality useful for a controlled flooding or topology discovery. Its 8-bit counter is incremented whenever the originator sends a new broadcast or multicast packet. The scope of the *fragmentation header* is to break datagrams whose payload does not fit into a single frame and dispatch them. If the entire payload fits in a single frame, no fragmentation header should be contained in the LoWPAN frame.

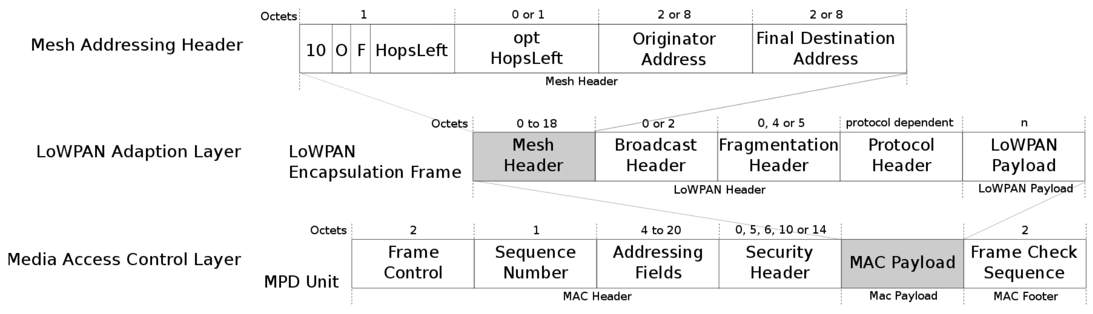


Figure 7.2: Frame structure of the SunSPOTs

Together with the protocol header and the actual payload from the network layer above, the LoWPAN encapsulation frame is delivered to the MAC layer below. Here it is added as MAC payload together with sequence number, address fields etc, into the MAC Protocol Data (MPD) for further processing. In order to group the relevant layers into the context, the functions which are specified and verified are highlighted in Figure 7.2.

Implementation

The implementation of the LoWPAN adaption layer consists of an interface and provides classes `LowPan`, `LowPanHeader`, `LowPanHeaderInfo`, and `LowPanPacket`. It is implemented in the following way: Class `LowPan` provides the data service of the LoWPAN adaption layer, encapsulated into an object of class `LowPanPacket`. This class is responsible for the correctness of the LoWPAN structure of each encapsulation frame. In addition, it provides the interfaces that support correct reading and writing to these objects. The class `LowPanHeader` collects header information that is necessary to create and send LoWPAN encapsulation frames.

In this sense, all information from the network layer can be written within one step since all information about optional header is already present at this point. In class `LowPanHeaderInfo` all header information of the LoWPAN encapsulation frame is stored after the successful reception of a packet and decapsulation of its MAC payload.

7.3 Verification of the Network Library

As mentioned before we investigate the send procedure from the networking library. These are essentially the classes that belong to the LoWPAN adaption layer and include functionality for fragmentation, and packaging. Important classes for the verification task are the classes `LowPan`, `LowPanPacket`, and `LowPanHeader`. For verification of other classes and their interfaces not considered here we refer to the work of [Sch08].

In addition to the above mentioned classes, auxiliary methods are needed that realize additional functionality and implement the radio packets. They are realized through the classes `RadioPacket`, and `Utils` which provide methods for arithmetic calculations and operations on byte arrays. The number of methods and attributes which are considered during the verification process is illustrated in Table 7.1. As shown in the table, many of the considered methods are relatively small. Only about 20 of the treated methods have

	LowPan	LowPanHeader	LowPanPacket	RadioPacket	Utils
methods 1-2 loc	6	27	6	31	26
methods 3-8 loc	6	1	28	27	8
methods 9-25 loc	6	1	16	2	0
methods $\geq$26 loc	13	0	6	1	0
total number of methods	31	29	56	61	34
number of attributes	18	32	31	33	3

Table 7.1: Number of methods and attributes of the classes involved in the verification process.

26 or more lines of code (loc).

What is in this context meant by verification is a multistage approach which consists of two steps. At the beginning a specification written in JML is added to each method. Since the specifications must be inferred from the software, RFCs and other available sources, this is an error prone and time intensive task. Afterwards, in the second step they are checked against their implementation using the theorem prover KeY. In case that the software matches the specification, they are considered as verified.

The verification strategy is chosen in a bottom-up manner which stipulates that firstly all methods that involve no other method are specified and verified. In addition, also methods which are trivial to specify are considered in this step. When doing so simple methods that are easy to verify are treated first and later on more complex methods — that make use of already verified methods — can be proved. In this way a verification order is obtained starting with the simple methods that have no dependencies and later on verify highly dependent functions.

7.4 Results

In this section the findings of the verification work with the KeY tool are presented. A short overview of the actual state of affairs when considering the verification of the network library is given in the consecutive Subsection 7.4.1. This is more or less only a general overview of the previous work. A more detailed investigation is started afterwards where a specific method is picked and proven. As an example method `getHopsLeft()` from the `LowPanPacket` class is considered and verified in Subsection 7.4.2.

7.4.1 The Networking Library

No remarkable results like software errors or inconsistencies of the code from the networking library were found. This can be either due to the fact that there exist none, or caused simply by missing or wrong specifications. Still a third option with respect to the size of proofs exists. Since some of the proofs could not be closed on a standard PC with 3 GB of

RAM we are not able to give any statements about them. So it is unclear whether the code of these methods meet their respective specifications or not.

The mechanism that covers the sending of a packet involves about 100 methods. As it turned out, much of the time spent was for house-keeping reasons like proof management, and problems with the KeY tool like the excessive use of bit operations. Due to this fact not all of the verifications goals were accomplished. So eventually we were able to verify approximately 80 of 100 methods. In addition much effort is spent on specifications added to the methods. These specifications which are expressed in the JAVA Modeling Language JML (see Section 2.3.5 on page 20) are annotated as comments to the code and sum up to 1 486 lines.

For the treated classes approximately 351 proofs exist. The largest ones have approximately 182 000 proof steps. These proofs will not be discussed here in detail. For a complete overview of the treated verification task we refer to [Sch08]. In the following unpublished verification goals are illustrated.

7.4.2 A detailed Example

In the following part, the formal correctness of method `getHopsLeft()` and its verification with the KeY environment is considered. The method's source code and its JML annotations are depicted in Figure 7.3. For each packet the method is invoked and computes the number of hops left for a packet by decrementing the counter and writing it to packets for dispatching as explained earlier in Section 7.2.1. If the hop counter exceeds 0, this packet is dropped.

Specification

The considered method has one contract and is specified as follows:

The pre-condition is formulated by the use of two `requires` clauses which state that the radio packet `rp` must be initialized and its payload must have a positive offset (≥ 0). Hence it is guaranteed that only valid packets are considered.

The post condition has two `ensures` clauses and stresses that if the field `HopsLeft` from the address field in the mesh header is equal `0xF` (see Figure 7.2), the extended hop counter is used. Remember that it is found in the 8-bit field `opt.HopsLeft` which follows the 4-bit `HopsLeft` field in the mesh header. In case that variable `extendedHops` equals `true`, then the result is a bit-wise conjunction of the MAC payload of packet `rp` with `0xFF`. Otherwise a bit-wise `and` is used to compute the results of `HOPSLEFT_BIT` that is `0x0F` and the `HopsLeft` variable. The purpose of this method is to use bit-stuffing to fill up the 4-bit hop counter to a byte.

By the specification of `normal_behavior` a normal termination of the method is expected as a further post condition. The clause `assignable \nothing;` demands that no location of the method should be altered during the call.

Verification

The basic proof obligations that need to be considered for every method are defined by Lemma 8.10 from [BHS07]. We consider S to be a specification for a program P and show the correctness of P with respect to S by proving the following three major proof

JAVA + JML

```
/*@ public normal_behavior
  @   requires    rp.offsetsInitialised;
  @   requires    0 <= rp.payloadOffset;
  @   ensures          extendedHops
  @   ==> \result ==  (byte) (getRadioPacket().getMACPayloadAt(hopsLeftIndex)
                                     & 255);
  @   ensures          !extendedHops
  @   ==> \result ==  (byte) (getRadioPacket().getMACPayloadAt(hopsLeftIndex)
                                     & HOPSLEFT_BITS);
  @   assignable  \nothing;
  @*/
public /*@ pure @*/ byte getHopsLeft() {
        byte hopsLeft = rp.getMACPayloadAt(hopsLeftIndex);
        if (extendedHops)
                return (byte) (hopsLeft & 0xff);
        else
                return (byte) (hopsLeft & HOPSLEFT_BITS);
    }
```

JAVA + JML

Figure 7.3: Sources and JML annotation of function `getHopsLeft()`.

obligations: the insurance that the method fulfills its post condition (EnsuresPost), the RespectsModifies meaning that only the denoted locations of the methods are altered, and finally the PreservesInv condition stating that all invariants are preserved by the execution of the method.

In the first step, the verification of *EnsuresPost* of method `getHopsLeft()` takes 6 693 steps and the proof splits into 99 branches. By steps we mean the deductive application of rules to the problem. Essentially this means that all proof goals can hereby be closed. The proof for the *RespectsModifies* can be lead in 1 478 steps including 24 branches. The last proof obligation that considers preserved invariants is trivially true since the considered method is pure and the following Lemma holds (see Lemma 8.14 from [BHS07]):

Lemma 7 *If all operation contracts opct applicable to an operation op have an empty modifies clause and* $\models RespectsModifies(opct)$ *then op preserves every invariant.*

With this respect a complete correctness proof is given and method `getHopsLeft()` is verified with the use of KeY.

7.4.3 General Problems & Results

Since some of the verification steps of the SunSPOT's network libraries ran into problems, we shortly discuss these in the following.

KeY

```
estimateOrJint256 {
    \find(orJint(x, y) >= 256)
    \sameUpdateLevel
    \replacewith(moduloInt(x) >= 256 | moduloInt(y) >= 256)
    \heuristics(userTaclets1)
};
```

KeY

Figure 7.4: KeY tacklet for the Lemma `estimateOrJlong256`.

Bit Operations

In the software that in run on embedded systems bit operations are excessively used, e.g. the networking layer heavily uses them for the computation of network masks, header information and so on. The use of bitwise `and`s and `or`s lead to strongly splitting proofs. This could partly be compensated by the use of lemma that turned out to be powerful enough to handle all bit operations that occurred in this case study. In principle these lemmas allow the term substitution like:

- commutation of bit operations, i.e., `and(a,b)`$\rightsquigarrow$`and(b,a)`
- fusion of bit operators, i.e., `and(and(a,b),b)`$\rightsquigarrow$`and(a,b)`
- estimation of bit operators, i.e., to estimate 8 bit integer values they are resolved in KeY with the help of Lemma `estimateOrJlong256` from Figure 7.4:

  `OrJint256(x,y)`
  $\rightsquigarrow$ `moduloLong(x) >=256 | moduloLong(y) >= 256`
  $\rightsquigarrow$ `((x+`2^{63}`) % `2^{64}` - `2^{63}`) >= 256 | ((y+`2^{63}`) % `2^{64}` - `2^{63}`) >= 256`

What is important to know is that 120 new lemmas are added to make the verification with KeY feasible. Some of the lemma are easy to check where others exists, which do not seem correct at first glance. Nevertheless there is no correctness proof of these lemma used during the deductive verification.

Strategy "Contracts"

For verifying methods with more than one contract the strategy was arbitrary chosen by the KeY prover. In consequence the choice was in many cases insufficient in the context of the strategy, causing proofs that were not optimal and sometimes even absurd. Further, the strategy "Contracts" was using the invariant of a class where a methods contract is defined and was set as assumed-selection. Although this can be a very useful strategy, corrupted proofs could be obtained using such a proof system. Fortunately this has been fixed in the KeY tool which now uses a combination of all applicable pre-conditions

Other related problems

During the verification work we noticed problems with a more or less strong impact on the considered case study. In this sense to name is the problem with the representation of number ranges in JML and KeY. Where specifications in JML use a JAVA like number representation, KeY uses a mathematical model with infinitely large numbers. When translating the JML specification to JAVADL theses numbers cannot be mapped correctly. This can be solved by adding special invariants to the requires clauses that incorporate a JAVA-like view on the treatment of numbers.

Other problems that often appear are strongly splitting proofs caused by programs with many method calls, or by a heavy use of `if`-cascades that excessively branch the proof. Even complicated specifications can increase the complexity of proofs that cannot be closed in consequence due to memory restrictions on the host system.

7.5 Conclusion

This chapter gives a brief overview of the application of the KeY tool to the SunSPOT's networking library and to classes of the LoWPAN adaption layer. In the verification the built-in dispatch mechanism that the user can access using `send()` is formally proven. The work is accomplished in two steps. As a first task the specification of relevant methods in JML is considered. Secondly, proof obligations are verified that ensure the correctness of post conditions, the respect of modifies clauses, and the preservation of invariants. Eventually we were able to verify a large part of the library, and only physical constraints like the computer's memory size hindered use from proving more.

Although some challenges still exist, the here presented approach is promising for future projects that use proof checking on a commercial object oriented software. Furthermore, the work of [Sch08] gives a comprehensive overview about other important aspects and problems including specifications and their subsequent verification.

Up to this point we applied many different tools and formalisms to sensor node related topics, covering many aspects and methods. The following and last chapter summarizes these results, that were obtained by investigating wireless sensor networks with practices and tools from the formal method community. As some of the results could also influence future work, we give suggestions for possible directions which promise to be interesting for subsequent work in the following conclusion.

CHAPTER 8

Conclusion & Thoughts about Future Work

This thesis covers the applicability of formalisms from the formal methods field to the area of wireless sensor nodes, their software, protocols and transport mechanisms. In this sense, after the fundamental basis is defined and the technical principles are introduced, investigations start in different directions. In principle, we can identify in this work two building blocks that capture interesting aspects of sensor networks in a broader sense.

The first block is covered in Chapter 3 and 4. Here, the objective is to analyze the efficiency aspects and energy related topics which are meaningful since sensor nodes are limited in energy and consequently have to operate energy-conscious and efficient. We gave proof that for this type of experiments, a variety of formalisms from logics exist, to obtain a suitable modeling, abstraction, and analysis.

The second building block considered in Chapters 5 to 7 provides correctness considerations of protocols and algorithmic aspects. This area of research shows that for many tasks appropriate models with the required level of abstraction exist. Nevertheless, without an adequate level of abstraction, the complexity is growing rapidly and further verification is no longer possible.

Due to the fact that all of the following chapters are almost independent of each other since they pursue research in sensor network from different point of views, suggestions for future work are contained after the conclusion of each individual chapter. In this sense, we investigate in the third chapter the energy use for sensor nodes and different routing strategies in a fixed but flexible topology.

Chapter 3: Energy Efficient Routing and Scheduling. In Chapter 3 we analyze the applicability of search algorithms from the model checking area and their practical use in the domain of wireless sensor networks. The computation of energy related results for sensor nodes, and routing devices for the packet transport from the sensor nodes down to the base station is in focus. On the basis of timed automata a node model is designed in UPPAAL which is capable of collecting sensor information, and send data using multi-hop in a flexible topology.

Since many of these sensor node models are combined, their concurrent execution can be analyzed with respect to certain constraints like common safety and liveness properties that guarantee the proper function of the network. In addition, energy related questions are analyzed by integrating energy specific power draws into the model. By this combination of timed automata models and sensor networks, we are able to answer energy related questions like "what is the best sending strength to choose, causing the fewest collisions?"

We belief that further investigations in this field are possible but would not gain any further insights. What might be interesting is to check whether new releases of the used tools allow the computation of actual timing information involved in the energy spending. In the example at hand, no values for the product of time and the energy spent in a state could be obtained automatically due to variable type constraints and therefore these values were computed manually.

In the current chapter the packet transport from the leaves to the sink is treated. We did not consider the opposite direction, namely the required energy for distributing information from the sink to the leaves, and start with this study in the following chapter.

Chapter 4: Performance Evaluation of Probabilistic Flooding Protocols. We analyze a broadcasting mechanism for authenticated queries and determined the effectiveness of different input parameters. Here we use a Markov model to give precise energy draws as opposing to simulations, and show that the choice of the parameters and security strongly depend on the considered topology. In addition, the tradeoff between energy and security is described by a function, showing optimal reference values for different scenarios. Since sensor nodes may run out of energy, the effectiveness of the protocol is pinpointed for different parameter settings that directly correspond with the energy draw and the involved authenticity of packets.

In future work, an extension of the modeled protocol is thinkable that also accounts for undecidable queries, i.e., queries for which nodes cannot decide whether they carry a valid authenticator or not. These are essentially packets equipped with an authenticator for which a node does not have a single matching key. In the current considered approach the queries are forwarded, but it is also worth to check if nodes only forward them sporadically and hereby efficiently encounter battery draining attacks. With the use of such a model the quality of results could be further increased.

This chapter concludes the energy analysis of this thesis, considered now from both directions: In the previous chapter from sensor nodes to the sink, and in the current chapter in the opposite direction. In the subsequent chapters a new part is started which now considers the verification and correctness of protocols and embedded software by formal methods. Until now such properties only played a minor role.

Chapter 5: Formal Verification of Probabilistic Query Dissemination. A novel protocol is investigated for authenticated query dissemination in the field of sensor networks with respect to several properties. It turns out that the protocol's underlying algorithm is exceptionally useful for small networks since the overhead that is spent for authenticity is minimal with respect to energy. Since this approach is probabilistic, the verification task is more elaborate. To enable a quantitative comparison of the proposed protocol, a compact recursive formula capturing the efficiency of the algorithm is derived. It's correctness is proved using formal methods. Additionally liveness and safety criteria that guarantee a proper operation of the protocol are provided. Furthermore the safety analysis for the AQF protocol is given, showing the probability that node and base station do not share even a single key.

In this chapter no topic arose that would be worth a future investigation. This is mainly the case since all relevant properties are checked. Of course, the models and conclusions can be consulted for investigations which consider similar probabilistic protocols or in detail cover a probabilistic authenticator checking.

The mere fact that the results of the model correspond to the derived probability formula still give no exhaustive guarantee for its correctness. Even if unlikely, we have no guarantee that these models are correct since the are developed by hand. Whenever human intervention is involved, there is the potential possibility of errors, also in the present situation. With regard to this, an automated model generation has many advantages, but in turn models become complex since the abstraction level is reduced or even completely removed. For this reason we provide in the following chapter a generation mechanism which translates sensor node implementations into software behavior models that can be used for verification.

Chapter 6: A Verifiable and Secure Concast Protocol We present a formal analysis of a concast protocol and analyze two different aspects. The one aspect covers the correctness based on a manually created model which is used to verify protocol related properties with SPIN, where especially the distributed character plays a role. It basically has the immense advantage that the modeled abstraction level can be independently chosen by the user, and hence the model's complexity is relatively low. But manual model creation is annoying as it has many drawbacks. In addition, there can be aspects of the ESAWN protocol which are modeled wrong since SPIN does not have appropriate formalisms to correctly reflect the intended behavior. In this sense, we cannot conclude the protocol's correctness by verifying a manually created artefact. Hence a more realistic model is needed that is derived using almost no manual interference.

The second approach fulfills this criterion. Here, a new software model is almost fully automatically derived from a description of sensor nodes in TinyOS, a development platform for sensor network software. Special about this model is, that it represents the behavior of a full functional sensor node, defined in ANSI-C. In addition, the model has no level of abstraction since it represents the real node behavior.

It turns out for the software model that the main problem is caused by the bounded model checker and its handling of the sending and receiving procedure. Since ESAWN heavily depends on message passing between nodes, CBMC is not able to handle the sophisticated properties that cover the correct treatment of ESAWN packets. For the basic properties that cover the broadcast of the initial configuration, specifications are successfully checked.

Many of the main difficulties with the C Bounded Model Checker were caused due to complex sending and receiving routines, which also involve the handling of sending queues. It is possible that other existing tools can treat theses mechanisms better, although we did no check on that. Instead we hope that a new release of CBMC will implement the missing formalisms and, in particular, enable the treatment of more sophisticated properties.

Special about this chapter is that many directions exist to proceed with the present work.

Due to the fact that we were not able to verify all specifications due to the missing implementation in the verification tool CBMC, this could be kept as a further task, although it is not very challenging. By the use of SlicX [PK07], additional specifications which cover temporal properties can be added and checked in a future step. It is also possible to test the generated software model and the used verification tool by fault injection. This was done for one property, but can likely be repeated for others.

An additional point of interest is also the following: A software behavior model in TinyOS can never fully describe a real device behavior since it lacks a hardware model and in consequence the resulting concurrent nature of these devices is not available. Two problems are involved with this: First of all, a verifier which is capable of concurrent systems would be necessary. This is accomplished by bounded model checking of concurrent programs with threats, investigated in the work of [RG05]. Secondly, a model description of the hardware would be required which is then run concurrent to the software.

In a further research topic, a modification for the NULL platform which was used for the model generation could be implemented, with some basic means for packet handling, especially the sending and receiving. This would make the manual modifications completely obsolete. Furthermore, it is desirable for such a platform to introduce appropriate abstraction for non-deterministic or random behavior by over approximation. But most important, by using an appropriate platform description there are no limits on the verification of software written in TinyOS since all existing implementations can be easily derived into ANSI-C. This would have a tremendous impact on the sensor network community because developers of software can now easily check the correctness of provided specifications within only a few steps.

All of the work in this chapter only considers automated software generation from the TinyOS platform. Another software environment is available in JAVA for the implementation and deployment of SunSPOT networks which is covered in the following chapter.

Chapter 7: Survey of the Sun SPOTs Networking Library In this chapter a theorem prover called KeY is applied to the SunSPOT's Networking Library, a commercial product with publicly available software sources. By the specification of properties in JML, about 80 of 100 methods of the network library implementing the LoWPAN adaption layer are verified. It turns out that the remaining 20 methods could not we proved mainly due to the heavy use of bit operations that cause an exponential blow-up by branching. Still, the here presented work shows the progress of deductive theorem provers and in particular the usability of KeY.

Of course there always exists the challenge to prove also the remaining parts of the SunSPOT's networking library. But this would require not only valid specifications which are very time consuming to derive without the knowledge of the software engineer, and the later proof of such a huge system is also very work intensive.

In summary, we showed in the present work the applicability of techniques known from the area of formal methods, and applied them to aspects from sensor networks, in particular wireless sensor networks.

Bibliography

[AH99] Rajeev Alur and Thomas A. Henzinger. Reactive Modules. In *Formal Methods in System Design*, pages 7–48, 1999.

[ASSB96] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying Continuous Time Markov Chains. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 269–276. Springer, July 1996.

[Atm] Atmel Corp. *ATmega128 datasheet*, atmel document no. 2467 edition. `http://www.atmel.com/dyn/resources/proddocuments/doc2467.pdf`.

[BAN90] Michael Burrows, Mart Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8:18–36, 1990.

[BB08] Erik-Oliver Blass and Zinaida Benenson. Das ZeuS-Angreifermodell. Technical Report TM-2008-1, Institut für Telematik, February 2008. ISSN 1613-849X.

[BBHM05] Gerd Behrmann, Ed Brinksma, Martijn Hendriks, and Angelika H. Mader. Scheduling lacquer production by reachability analysis - A case study. In P. Horacek, M. Simandl, and P. Zitek, editors, *16th IFAC World Congress, Prague, Czech Republic*, page Mo_A17_TO/3, Laxenburg, Austria, July 2005. International Federation of Automatic Control (IFAC).

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer Verlag, 2004.

[BCF07] Zinaida Benenson, Peter M. Cholewinski, and Felix C. Freiling. *Vulnerabilities and Attacks in Wireless Sensor Networks*, volume 1 of *Cryptology & Information Security Series (CIS)*, chapter Vulnerabilities and Attacks in Wireless Sensor Networks. IOS Press, April 2007.

[BCHG99] Christel Baier, Edmund M. Clarke, and Vasiliki Hartonas-Garmhausen. On the Semantic Foundations of Probabilistic Synchronous Reactive Programs. *Electronic Notes in Theoretical Computer Science*, 22:3 – 28, 1999. PROBMIV'98, First International Workshop on Probabilistic Methods in Verification.

[BCZ05] Erik-Oliver Blaß, Michael Conrad, and Martina Zitterbart. A Tree-Based Approach for Secure Key Distribution in Wireless Sensor Networks. In *REAL-WSN – Workshop on Real-World Wireless Sensor Networks*, Stockholm, Sweden, June 2005.

[BdA95] A. Bianco and L. de Alfaro. Model Checking of Probabilistic and Nondeterministic Systems. In P. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[BDL04] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[Bec01] Bernhard Beckert. A Dynamic Logic for the Formal Verification of Java Card Programs. In I. Attali and T. Jensen, editors, *JavaCard '00: Revised Papers from the First International Workshop on Java on Smart Cards: Programming and Security*, volume 2041 of LNCS, pages 6–24, London, UK, 2001. Springer-Verlag.

[Ben08] Zinaida Benenson. *Access Control in Wireless Sensor Networks*. PhD thesis, Universität Mannheim, 2008.

[BFH$^+$06] Zinaida Benenson, Felix C. Freiling, Ernest Hammerschmidt, Stefan Lucks, and Lexi Pimenidis. Authenticated Query Flooding in Sensor Networks. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0:644–647, 2006.

[BH05] Richard Bubel and Reiner Hähnle. Formal Specification of Security-Critical Railway Software with the KeY System. *Software Tools for Technology Transfer*, 7(3):197–211, June 2005.

[BHJM07] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The Software Model Checker Blast. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(5-6):505–525, 2007.

[BHS07] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*. LNCS 4334. Springer-Verlag, 2007.

[BISV08] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, and Martin Vetterli. The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments. In *The 6th ACM Conference on Embedded Networked Sensor Systems (SenSys 2008)*, pages 43–56, 2008.

[BKH99] Christel Baier, Jost-Pieter Katoen, and Holger Hermanns. Approximate Symbolic Model Checking of Continuous-Time Markov Chains. In J. Baeten and S. Mauw, editors, *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161, Eindhoven, August 1999. Springer.

[BLR05] Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Optimal scheduling using priced timed automata. *SIGMETRICS Perform. Eval. Rev.*, 32(4):34–40, 2005.

[BLS04] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In *CASSIS 2004, LNCS vol. 3362*, pages 49–69. Springer, 2004.

[BWZ08] Erik-Oliver Blaß, Joachim Wilke, and Martina Zitterbart. Relaxed authenticity for data aggregation in wireless sensor networks. In *SecureComm '08: Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pages 1–10, New York, NY, USA, 2008. ACM.

[BY04] Johan Bengtsson and Wang Yi. Timed Automata: Semantics, Algorithms and Tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124. Springer Berlin / Heidelberg, July 2004.

[CDE08] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In Richard Draves and Robbert van Renesse, editors, *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*, pages 209–224. USENIX Association, 2008.

[CE82] Edmund M. Clarke and Allen E. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

[CGSW99] Kenneth L. Calvert, James Griffioen, Amit Sehgal, and Su Wen. Concast: Design and implementation of a new network service. In *In Proceedings of 1999 International Conference on Network Protocols*, 1999.

[Chi07] Chipcon. *CC2420 manual: 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B)*, March 2007. `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`.

[CKL04] Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. A Tool for Checking ANSI-C Programs . In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

[CKY03] Edmund M. Clarke, Daniel Kroening, and Karen Yorav. Behavioral Consistency of C and Verilog Programs Using Bounded Model Checking. Technical Report CMU-CS-03-126, School of Computer Science, Carnegie Mellon University, 2003.

[CLQR07] Shaunak Chatterjee, Shuvendu Lahiri, Shaz Qadeer, and Zvonimir Rakamaric. A Reachability Predicate for Analyzing Low-Level Software. In Springer Verlag, editor, *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 07*, volume MSR-TR-2006-154. Microsoft Research, April 2007.

[CMST08] Ernie Cohen, Michał Moskal, Wolfram Schulte, and Stephan Tobies. A Precise Yet Efficient Memory Model For C. unpublished, October 2008.

[Cro05] Crossbow Inc. *Motes, Smart Dust Sensors, Wireless Sensor Networks*, 2005. `http://www.xbow.com/Products/productdetails.aspx?ais=3`.

[CT] Inc. Crossbow Technology. MicaZ data sheet.

[CTTV04] Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by Network Decomposition. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK*, volume 3170 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2004.

[CTW99] M. Chaudron, J. Tretmans, and K. Wijbrans. Lessons from the Application of Formal Methods to the Design of a Storm Surge Barrier Control System. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – World Congress on Formal Methodsin the Development of Computing Systems II*, volume 1709 of *Lecture Notes in Computer Science*, pages 1511–1526. Springer-Verlag, 1999.

[Dij72] Edsger W. Dijkstra. Notes on structured programming. In O.-J. Dahl, E.W. Dijkstra, , and C.A.R. Hoare, editors, *Structured Programming*, pages 1–81, New York, 1972. Academic Press.

[ES03] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[FG06] Ansgar Fehnker and Peng Gao. Formal Verification and Simulation for Performance Analysis for Probabilistic Broadcast Protocols. In *5th International Conference on AD-HOC Networks & Wireless*, volume 4104 of *LNCS*, pages 128–141. Springer, 2006.

[Fru06] M. Fruth. Probabilistic Model Checking of Contention Resolution in the IEEE 802.15.4 Low-Rate Wireless Personal Area Network Protocol. In *Proc. 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'06)*, 2006.

[FvHM07a] Ansgar Fehnker, Lodewijk F. W. van Hoesel, and Angelika H. Mader. Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks. In J. Davis and J. Gibbons, editors, *Proceedings of the 6th International Conference on Integrated Formal Methods, IFM 2007, Oxford, Britain*, volume 4591 of *Lecture Notes in Computer Science*, pages 253–272, Berlin / Heidelberg, June 2007. Springer Verlag.

[FvHM07b] Ansgar Fehnker, Lodewijk F. W. van Hoesel, and Angelika H. Mader. Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks. In J. Davis and J. Gibbons, editors, *Proceedings of the 6th International Conference on Integrated Formal Methods, IFM 2007, Oxford, Britain*, volume 4591 of *Lecture Notes in Computer Science*, pages 253–272, Berlin / Heidelberg, June 2007. Springer Verlag.

[Han07] Youssef Hanna. SLEDE: lightweight verification of sensor network security protocol implementations. In *ESEC-FSE companion '07: The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*, pages 591–594, New York, NY, USA, 2007. ACM.

[HCL08] Gregory Hackmann, Octav Chipara, and Chenyang Lu. Robust topology control for indoor wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 57–70, New York, NY, USA, 2008. ACM.

[HCSY92] Thomas A. Henzinger, Xavier Cicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *Proc. 7th Annual IEEE Symposium on Logic in Computer Science*, pages 394–406, 1992.

[HJ94] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[HKNP06] A. Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In Holger Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Journal of Information and Computation*, 111(2):193–244, 1994.

[Hol03] Gerard J. Holzmann. *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley Professional, September 2003.

[HS00] Gerard J. Holzmann and Margaret H. Schmith. Automating Software Feature Verification. Technical report, Bell Labs Technical Journal, 2000.

[Kal92] B. Kaliski. RFC 1319: The MD2 Message-Digest Algorithm. Technical report, RSA Laboratories - Network Working Group, April 1992.

[Kat03] Jost-Pieter Katoen. *Principles of Model Checking*. University of Twente, Formal Methods and Tools Group, lecture nodes edition, 2002/2003.

[KDD04] Uwe Kubach, Christian Decker, and Ken Douglas. Collaborative control and coordination of hazardous chemicals. In *2nd international conference on Embedded networked sensor systems 2004 (SenSys 2004)*, November 03 - 05 2004.

[KMG08] N. Kothari, T. Millstein, and R. Govindan. Deriving State Machines from TinyOS Programs Using Symbolic Execution. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 271–282, April 2008.

[KNP01] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic Symbolic Model Checker. In P. Kemper, editor, *Proc. Tools Session of Aachen 2001 International Multiconferenceon Measurement, Modelling and Evaluation of Computer-CommunicationSystems*, pages 7–12, September 2001. Available as Technical Report 760/2001, University of Dortmund.

[KNP05] Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic Model Checking and Power-Aware Computing. In *Proc. 7th International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS'05)*, pages 6–9, 2005.

[KNP07] Marta Kwiatkowska, Gethin Norman, and D Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM 2007)*, volume 4486 of *LNCS (Tutorial Volume)*, pages pages 220–270. Springer, 2007.

[KNS02] Marta Kwiatkowska, Gethin Norman, and J. Sproston. Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol. In Holger Hermanns and R. Segala, editors, *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, pages 169–187. Springer, 2002.

[KNSW04] Marta Kwiatkowska, Gethin Norman, J. Sproston, and F. Wang. Symbolic Model Checking for Probabilistic Timed Automata. In Y. Lakhnech and S. Yovine, editors, *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant Systems (FORMATS/FTRTFT'04)*, volume 3253 of *LNCS*, pages 293–308. Springer, 2004.

[lab04] Intel Berkeley Research lab. Intel Lab Data. `http://db.csail.mit.edu/labdata/labdata.html`, April 2004.

[LBB$^+$01] Kim G. Larsen, Gerd Behrmann, Ed Brinksma, Ansgar Fehnker, Thomas Hune, Paul Pettersson, and Judi Romijn. As Cheap as Possible: Efficient Cost-Optimal Reachability for Priced Timed Automata. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of CAV 2001*, number 2102 in Lecture Notes in Computer Science, pages 493–505. Springer–Verlag, 2001.

[LBR98] Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary design of JML: A behavioral interface specification language for Java. Technical Report 98-06, Iowa State University, Department of Computer Science, 1998.

[LL03] Philip Levis and Nelson Lee. *TOSSIM: A Simulator for TinyOS Networks*, 1.0 edition, September 2003.

[LLWC03] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys '03)*, pages 126–137, New York, NY, USA, 2003. ACM.

[LO05] Ciaran Lynch and Fergus O'Reilly. Processor Choice for Wireless Sensor Networks. In *Proc. 1st Workshop on Real-World Wireless Sensor Networks REALWSN*, number T2005:09 in SICS Technical Reports, pages 58–62. SICS, Stockholm, Sweden, 2005.

[LPC$^+$08] Gary T. Leavens, Erik Poll, Curtis Clifton, Yoonsik Cheon, Clyde Ruby, David Cok, Peter Müller, Joseph Kiniry, Patrice Chalin, and Daniel M. Zimmerman. *JML Reference Manual*, 1.235 (draft) edition, May 2008.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1997.

[Mar97] Dowson Mark. The Ariane 5 software failure. *SIGSOFT Softw. Eng. Notes*, 22(2):84, 1997.

[McI06] A. K. McIver. Quantitative refinement and model checking for the analysis of probabilistic systems. In *Proc. Formal Methods (FM 2006)*, volume 4085 of *LNCS*, pages 131–146. Springer Verlag, 2006.

[MFHH05] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, March 2005.

[MKHC07] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. Network Working Group. Internet-Draft, February 2007.

[Möl96] Karl-Heinz Möller. Ausgangsdaten für Qualitätsmetriken - Eine Fundgrube für Analysen. *Software-Metriken in der Praxis*, 1996.

[Mos07] Wojciech Mostowski. Fully Verified Java Card API Reference Implementation. In Bernhard Beckert, editor, *Verify'07 4th International Verification Workshop*, volume 259 of *CEUR WS*, July 2007.

[Mot06] Moteiv Corporation. *Tmote Sky Datasheet*, 1.04 edition, November 2006. `http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf`.

[Net] *Network Simulator ns-2*. `http://nsnam.isi.edu/nsnam/`.

[NPK$^+$02] Gethin Norman, David Parker, Marta Kwiatkowska, S. Shukla, and R. Gupta. Formal Analysis and Validation of Continuous Time Markov Chain Based System Level Power Management Strategies. In W. Rosenstiel, editor, *Proc. 7th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'02)*, pages 45–50. IEEE Computer Society Press, 2002.

[NPK$^+$03] Gethin Norman, David Parker, Marta Kwiatkowska, S. Shukla, and R. Gupta. Using Probabilistic Model Checking for Dynamic Power Management. In M. Leuschel, S. Gruner, and S. Lo Presti, editors, *Proc. 3rd Workshop on Automated Verification of Critical Systems (AVoCS'03)*, Technical Report DSSE-TR-2003-2, University of Southampton, pages 202–215, April 2003.

[NTCS99] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, New York, NY, USA, 1999. ACM.

[Pau94] Lawrence C. Paulson. *Isabelle: a Generic Theorem Prover*. Number 828 in Lecture Notes in Computer Science. Springer – Berlin, 1994.

[Pim98] Kars Pim. Formal Methods in the Design of s Storm Surge Barrier Control System. In *Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems*, pages 353–367, London, UK, 1998. Springer-Verlag.

[PK07] Hendrik Post and Wolfgang Küchlin. Integrated static analysis for linux device driver verification. In Jim Davies and Jeremy Gibbons, editors, *IFM*, volume 4591 of *Lecture Notes in Computer Science*, pages 518–537. Springer, 2007.

[Pri] *Prism Manual*, 3.2 edition. `http://www.prismmodelchecker.org/`.

[QS82] J. P. Quielle and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *In Proceedings of the 5th International Symposium on Programming*, pages 337–350, 1982.

[RFC89] RFC 1122 standard. *Requirements for Internet Hosts—Communication Layers*. Internet Engineering Task Force (IETF), 1989.

[RG05] Ishai Rabinovitz and Orna Grumberg. Bounded model checking of concurrent programs. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2005.

[RKNP04] J.J.M.M. Rutten, Marta Kwiatkowska, Gethin Norman, and David Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, volume 23 of *CRM Monograph Series*. American Mathematical Society, 2004.

[RLS04a] Jacob Illum Rasmussen, Kim G. Larsen, and Kna Subramani. Resource-optimal scheduling using priced timed automata. In *10th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, number 2988 in LNCS, pages 220–235. Springer, 2004.

[RLS04b] J.I. Rasmussen, Kim G. Larsen, and K. Subramani. *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *LNCS*, chapter Resource-Optimal Scheduling Using Priced Timed Automata, pages 220–235. Springer, 2004.

[RNS00] Rustan, Greg Nelson, and James B. Saxe. ESC/Java User's Manual. Technical Note 2000-002, Compaq SRC, October 2000.

[Sch08] Chistoph Scheben. Verificatin of Sun SPOT's Network Library. Master's thesis, University of Karlsruhe (TH), Faculty of Computer Science, Institute of Theoretical Computer Science, May 2008.

[Soc06] IEEE Computer Society. IEEE std. 802.15.4 - 2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Technical report, IEEE Computer Society, September 2006.

[Soc07] IEEE Computer Society. IEEE Standard for Information technology, Telecommunications and information exchange between systems, Local and metropolitan area networks,Specific requirements. Technical Report 802.11-2007, IEEE Computer Society, June 2007.

[SOR93] N. Shankar, S. Owre, and J. M. Rushby. *The PVS Proof Checker: A Reference Manual*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.

[Spi09] *The model checker Spin*, 2009. `http://spinroot.com/spin/whatispin.html`.

[Sun08] Sun Labs. *Sun Small Programmable Object Technology (Sun SPOT): Theory of Operation*, August 2008.

[SW07] Peter H. Schmitt and Frank Werner. Model Checking for Energy Efficient Scheduling in Wireless Sensor Networks. Technical Report 2007-01, Universität Karlsruhe, January 2007.

[TCCP08] Llanos Tobarra, Diego Cazorla, Fernando Cuartero, and J. Jose Pardo. Modelling secure wireless sensor networks routing protocols with timed automata. In *PM2HW2N '08: Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, pages 51–58, New York, NY, USA, 2008. ACM.

[Ton07] Isabel Tonin. Verifying the Mondex Case Study.The KeY Approach. Techischer Bericht 2007-4, Fakultät für Informatik, Universität Karlsruhe, 2007.

[TXY08] Simon Tschirner, Liang Xuedong, and Wang Yi. Model-based validation of QoS properties of biomedical sensor networks. In *EMSOFT '08: Proceedings of the 7th ACM international conference on Embedded software*, pages 69–78, New York, NY, USA, 2008. ACM.

[Wei88] Mark Weiser. Ubiquitous Computing. `http://www.ubiq.com/hypertext/weiser/UbiHome.html`, 1988.

[Wei91] Mark Weiser. The Computer for the 21st Century. *Scientific American*, February 1991.

[Win05] A. Winter. *dB or not dB*. Rohde & Schwarz, application note 1ma98 edition, October 2005.

[WS07] Frank Werner and Peter H. Schmitt. Model Checking for Energy Efficient Scheduling in WSN. Technical report, 6. Fachgespräch Sensornetzwerke [GI/07], July 2007. Technischer Bericht der RWTH Aachen, Distrubuted Systems Group.

[WS08] Frank Werner and Peter H. Schmitt. Analysis of authenticated Query Flooding by Probabilistic Means. In *The 5th Annual Conference on Wireless on Demand Network Systems and Services (WONS 2008)*, pages 101–104, January 2008.

[WWB$^+$08] Joachim Wilke, Frank Werner, Markus Bestehorn, Zinaida Benenson, Simon Kellner, and Erik-Oliver Blaß. Vergleichbarkeit von Ansätzen zur Netzwerkanalyse in drahtlosen Sensornetzen. In Hartmut Ritter, Kirsten Terfloth, Georg Wittenburg, and Jochen Schiller, editors, *Fachgespräch Sensornetze - FGSN 2008*, number B 08-12 in Proceedings of 7. Fachgespräch Sensornetzwerke, pages 97–100, Berlin, Deutschland, September 2008. 7. Fachgespräch Sensornetzwerke.

[ZB06] Martina Zitterbart and Erik-Oliver Blaß. An Efficient Key Establishment Scheme for Secure Aggregating Sensor Networks. In *ACM Symposium on Information, Computer and Communications Security*, pages 303–310, Taipei, Taiwan, March 2006. ISBN 1-59593-272-0.

[ZBG98] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

[Zig05] ZigBee Alliance. *ZigBee Specification 1.0*, June 2005.

[ZXSJ03] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing pair-wise keys for secure communication in ad hoc networks: A probabilistic approach. George Mason University, Tech. Rep. ISE-TR-03-01, March 2003.

APPENDIX A

Energy Efficient Routing and Scheduling

A.1 Additional Functions for the Uppaal Analysis

The following functions were used for the efficiency analysis from Chapter 3. Function `CheckID()` is checking whether any device within range of device `id` (`dist[id][i]`) is able to receive in which case `true` is returned:

```
bool CheckID(id_t id){
    bool rv=false;
    int i=0;
    for (0; i<N; i++)
     if ((dist[id][i]<=TXpowerLow) & (a[i]==N))
         rv=true;
    return rv;
}
```

`Clean(id)` is resetting the channel for devices that carry the flag of sensor `id`.

```
void Clean(id_t id){
    int i=0;
    for (0; i<N; i++)
     if ((dist[id][i]<=TXpowerLow) & (a[i]==id))
         a[i]=−1;
}
```

Function `CleanAll(id)` clears the channel of all gadgets within the transmission range.

```
 void CleanAll(id_t id){
    int i=0;
    for (0; i<N; i++)
     if (dist[id][i]<=TXpowerLow)
         a[i]=−1;
}
```

Function `CheckAv(id)` checks the availability of each device within the transmission range by setting a flag on array `a[]`.

```
 void CheckAv(int id){
    int i;
    for (0; i<N; i++)
     if (dist[id][i]<=TXpowerLow)
```

```
        //Put Senders ID on Receivers Channel
        if (a[i]==-1) a[i]=id;
    else
        //Put Collision on Receivers Channel
        if(a[i]>-1) a[i]=N;
}
```

A.2 Various Constant Definitions

```
const int ProcActiveDraw=8000;              //8 mA
const int ProcSleepDraw =15;                //<15 muA
const int TransReceiveDraw = 19700;         //19.7 mA
const int TransMinus10dBmDraw = 11000;      //11 mA TX, -10 dBm
const int TransMinus5dBmDraw = 14000;       //14 mA TX, -5 dBm
const int Trans0dBmDraw  = 17400;           //17.4 mA TX, 0 dBm
const int TransIdleDraw  = 20;              //20 mu A, voltage regular on
const int TransSleepDraw = 1;               //1 mu A, voltage regulator off

//System Draw - States
const int PDown  = 2;         //15+1;
const int PSleep = 8;         //8000 + 1;
const int PIdle  = 8;         //8000 + 20;
const int PRcv   = 28;        //8000 + 19700;

//different TX strength
//const int PSnd   = 19;      //8000 + 11000, Sending with -10dBm
//const int PSnd   = 22;      //8000 + 14000, Sending with -5dBm
const int PSnd   = 25;        //8000 + 17400, Sending with  0dBm

const int ActivePeriod=1;       //Time Units a Mote stays active
const int ActiveCycle=100;      //Time units between 2 consecutive beacons
```

APPENDIX B

Analyzing Probabilistic Flooding

B.1 Prism Model of the Scenario 1

```
probabilistic
const double pf;

//Global Variables
//0 - Nothing received
//1 - received and deciding
global a : [0..1] init 1;
global b : [0..1] init 0;
global c : [0..1] init 0;
global d : [0..1] init 0;
global e : [0..1] init 0;
global f : [0..1] init 0;

module sensA
     //Local Variables
     //0 - Nothing received
     //1 - accepted query
     //2 - rejected query
     la: [0..2] init 0;

     [] (a=1) & (la=0) -> pf: (la'=1) & (b'=1) +
                          (1-pf): (la'=2);
endmodule

//Instantiate other modules by renaming
module sensB=sensA [la=lb, a=b, b=c] endmodule
module sensC=sensA [la=lc, a=c, b=d] endmodule
module sensD=sensA [la=ld, a=d, b=e] endmodule
module sensE=sensA [la=le, a=e, b=f] endmodule

//Reward Variables
rewards
     a=1: 1;
     b=1: 1;
     c=1: 1;
     d=1: 1;
     e=1: 1;
```

```
endrewards
```

B.2 Energy Computation for TMote Sky Nodes

```
use POSIX;

my $m = 100;             # authenticator size in bits
my $data = 8;            # data length in bits
my $keylen = 128;        # bits - alternatively 128 bit or 256 bit
my $l=1000;              # key pool size
my $k=50;                # keys per node
my $capNodes=1;          # captures nodes

my $pf;                  #probability of forwarding a faked packet
my $B;
my $bsnail;
my $matching_node_keys;
my $payload = $m + $data;     # bits - compute for every m

# Rounds running the MD2 cipher
my $md2_block_instr = 16 * 10 + 16 * 5 + 18 * (48 * 4 + 2);
my $mipms = 6 * 1000;

#Energy Costs for the TMote Sky node
my $pow_rx = 0.059;      # W
my $pow_tx = 0.056;      # W
my $pow_mv = 0.065;      # W
my $pow_cpu = 0.006;     # W

#accumulator variables
my $tot_nrg = 0;    # mJ
my $tot_tme = 0;    # ms
my $energy = 0;     # mJ
my $time = 0;       # ms

sub ComputePF(){
     #compute forwarding probability according to AQF
     $bsnail= $l*(1-(1- (($k/$l) ** $capNodes)));
     $B=$m*(($bsnail  + $l) / (2*$l));
     $matching_node_keys=($m*$k)/$l;
     $pf= ((($l-$k)/$l) + (($k*$B )/($l*$m))) ** $m;
}

sub ComputeEnergy(){
    ##Reset variables
    $tot_nrg=0;
    $tot_tme=0;

    ##CMP payload
    $payload = $m + $data;

    # packet reception
    $time = 0.5 + 0.035 * ($payload / 8.0);
    $energy = $pow_rx * $time;
    $tot_nrg += $energy; $tot_tme += $time;

    # packet loading
    $time = 2 + 0.05 * ($payload / 8.0);
    $energy = $pow_mv * $time;
    $tot_nrg += $energy; $tot_tme += $time;
    print "Energy␣(rcv):␣\t$tot_nrg\n";

    # hash of data
    $time = $md2_block_instr * floor((floor(($data+7)/8) + 31)/16) / $mipms;
    $energy = $pow_cpu * $time;
    $tot_nrg += $energy; $tot_tme += $time;

    # validating hashes
    $time = $matching_node_keys * $md2_block_instr
                * floor((floor(($data + $keylen + 7)/8) + 31) / 16) / $mipms;
    $energy = $pow_cpu * $time;
    $tot_nrg += $energy; $tot_tme += $time;
    print "Energy␣(RCV+CMP):␣\t$tot_nrg␣mJ\n";

    # packet sending
    $time = 2 + 0.05 * ($payload / 8.0);
    $energy = $pow_mv * $time;
    $tot_nrg += $energy; $tot_tme += $time;

    # packet transmission
    $time = 0.5 + 0.035 * ($payload / 8.0);
    $energy = $pow_tx * $time;
    $tot_nrg += $energy; $tot_tme += $time;
    print "Energy␣(RCV+CMP):␣\t$tot_nrg␣mJ\n";
}

    ComputePF();
    ComputeEnergy();
```

B.2.1 Energy for faked and legitimate queries

B.3 Prism Verification Results

B.3.1 Comparison of Prism and theoretical results

Probabilities for problem 2 (see Figure 4.1b) computed using PRISM by the formula that i nodes accept a query(ACC_i) with $p_f = 0.14$.

i	Markov chain	formula	difference
0	8.600000E-01	0.000000E+00	8.600000E-01
1	1.035440E-01	1.035440E-01	0.000000E+00
2	2.899232E-02	2.899232E-02	0.000000E+00
3	6.088387E-03	6.088387E-03	0.000000E+00
4	1.136499E-03	1.136499E-03	0.000000E+00
5	2.118381E-04	1.988873E-04	1.295080E-05
6	2.408849E-05	3.341307E-05	-9.324577E-06
7	2.592751E-06	5.457468E-06	-2.864717E-06
8	2.538357E-07	8.731949E-07	-6.193592E-07
9	2.066105E-08	1.375282E-07	-1.168671E-07
10	0.000000E+00	2.139327E-08	-2.139327E-08

B.3.2 Comparing different topologies

Energy in mJ for different network topologies. The first line indicated the power use for simple flooding without the AQF algorithm.

		$\ell = 1\,000$		$\ell = 5\,000$		$\ell = 10\,000$	
m	k	E_{fake}	$E_{nonfake}$	E_{fake}	$E_{nonfake}$	E_{fake}	$E_{nonfake}$
100	50	0.31	6.28	2.04	5.74	3.51	5.67
150	50	0.29	7.4	1.42	6.6	3.03	6.49
200	50	0.31	8.53	1.08	7.45	2.59	7.32
250	50	0.33	9.65	0.88	8.31	2.23	8.14
300	50	0.37	10.78	0.77	9.17	1.93	8.96
350	50	0.4	11.9	0.71	10.02	1.71	9.79
400	50	0.43	13.03	0.67	10.88	1.54	10.61
450	50	0.46	14.15	0.65	11.73	1.41	11.43
500	50	0.5	15.28	0.65	12.59	1.31	12.26
100	100	0.24	6.95	0.85	5.87	2.04	5.74
150	100	0.27	8.41	0.57	6.8	1.42	6.6
200	100	0.3	9.87	0.47	7.72	1.08	7.45
250	100	0.33	11.33	0.44	8.64	0.88	8.31
300	100	0.36	12.79	0.43	9.57	0.77	9.17
350	100	0.4	14.25	0.44	10.49	0.71	10.02
400	100	0.43	15.71	0.46	11.42	0.67	10.88
450	100	0.46	17.18	0.48	12.34	0.65	11.73
500	100	0.5	18.64	0.51	13.26	0.65	12.59
100	150	0.23	7.62	0.5	6.01	1.26	5.81
150	150	0.26	9.42	0.38	7	0.82	6.7
200	150	0.3	11.21	0.35	7.99	0.64	7.59
250	150	0.33	13.01	0.36	8.98	0.55	8.48
300	150	0.36	14.81	0.38	9.97	0.52	9.37
350	150	0.4	16.61	0.41	10.96	0.5	10.26
400	150	0.43	18.4	0.43	11.95	0.51	11.15
450	150	0.46	20.2	0.47	12.94	0.52	12.04
500	150	0.5	22	0.5	13.93	0.54	12.93
100	200	0.23	8.29	0.37	6.14	0.85	5.87
150	200	0.26	10.42	0.32	7.2	0.57	6.8
200	200	0.3	12.56	0.32	8.26	0.47	7.72
250	200	0.33	14.69	0.34	9.32	0.44	8.64
300	200	0.36	16.82	0.37	10.37	0.43	9.57
350	200	0.4	18.96	0.4	11.43	0.44	10.49
400	200	0.43	21.09	0.43	12.49	0.46	11.42
450	200	0.46	23.22	0.46	13.55	0.48	12.34
500	200	0.5	25.35	0.5	14.61	0.51	13.26
100	250	0.23	8.96	0.31	6.28	0.63	5.94
150	250	0.26	11.43	0.29	7.4	0.45	6.9
200	250	0.3	13.9	0.31	8.53	0.39	7.86
250	250	0.33	16.37	0.33	9.65	0.38	8.81
300	250	0.36	18.84	0.37	10.78	0.4	9.77
350	250	0.4	21.31	0.4	11.9	0.42	10.73
400	250	0.43	23.78	0.43	13.03	0.44	11.68
450	250	0.46	26.24	0.46	14.15	0.47	12.64
500	250	0.5	28.71	0.5	15.28	0.5	13.6

Table B.1: AQF Energy use for fake packets E_{fake} and legitimate packets $E_{nonfake}$.

m	MNKK	p_f	top 4	top 5	top 6	top 7	top 8
-	-	-	45.0306	32.4785	28.9645	24.7955	39.3432
100	1.6667	0.4392	19.0830	10.4167	10.5800	8.2174	13.6913
150	2.5000	0.2911	12.3856	7.3820	7.4834	6.3961	9.6075
200	3.3333	0.1929	9.3111	6.2246	6.2778	5.7378	7.6218
250	4.1667	0.1278	7.9657	5.8281	5.8565	5.5641	6.6920
300	5.0000	0.0847	7.4321	5.7782	5.7932	5.6232	6.3185
350	5.8333	0.0561	7.3082	5.9081	5.9158	5.8112	6.2531
400	6.6667	0.0372	7.4101	6.1420	6.1459	6.0788	6.3672
450	7.5000	0.0247	7.6469	6.4412	6.4430	6.3988	6.5910
500	8.3333	0.0163	7.9696	6.7835	6.7844	6.7546	6.8848

B.4 Prism Model for topology 4

```
//Prism Model topology 4
probabilistic

const double pf;
const double engr=42;          //in mu As     42
const double engrs=227.2;      //in mu As     185.2

//0 - Nothing received
//1 - received and deciding
global a : [0..1] init 1;
global b : [0..1] init 1;
global c : [0..1] init 1;
global d : [0..1] init 1;
global e : [0..1] init 0;
global f : [0..1] init 0;
global g : [0..1] init 0;
global h : [0..1] init 0;
global i : [0..1] init 0;
global j : [0..1] init 0;
global k : [0..1] init 0;
global l : [0..1] init 0;
global m : [0..1] init 0;
global n : [0..1] init 0;

module sensA
     la: [0..3] init 0;  //0 - Nothing,1 - accepted, 2 - rejected

     [] (a=1) & (la=0) ->     pf: (la'=1)&(b'=1)&(d'=1)&(c'=1)
                               + 1-pf: (la'=2);
     [ar]  (a=1) & (la=2) -> (la'=3);
     [ars] (a=1) & (la=1) -> (la'=3);
endmodule

module sensB
     lb: [0..3] init 0;  //0 - Nothing,1 - accepted, 2 - rejected

     [] (b=1) & (lb=0) ->     pf: (lb'=1)&(a'=1)&(d'=1)&(g'=1)&(e'=1)
                         + 1-pf: (lb'=2);
     [br] (b=1) & (lb=2) -> (lb'=3);
     [brs] (b=1) & (lb=1) -> (lb'=3);
endmodule
module sensC=sensB [lb=lc, b=c, a=a, d=d, g=h, e=f, br=cr, brs=crs] endmodule

module sensD
     ld: [0..3] init 0;  //0 - Nothing,1 - accepted, 2 - rejected

     [] (d=1) & (ld=0) -> pf: (ld'=1)&(a'=1)&(c'=1)&(h'=1) &(j'=1)&(g'=1)&(b'=1)
                              + 1-pf: (ld'=2);
     [dr] (d=1) & (ld=2) -> (ld'=3);
     [drs] (d=1) & (ld=1) -> (ld'=3);
endmodule


module sensF=sensA [la=lf,a=f,b=c,d=h,c=k,ar=fr,ars=frs] endmodule
module sensG=sensD [ld=lg,d=g,a=e,c=b,h=d,j=j,g=l,b=i,dr=gr,drs=grs] endmodule
module sensH=sensD [ld=lh,d=h,a=d,c=c,h=f,j=k,g=m,b=j,dr=hr,drs=hrs] endmodule
module sensI=sensA [la=li,a=i,b=e,d=g,c=l,ar=ir,ars=irs] endmodule
module sensK=sensA [la=lk,a=k,b=f,d=h,c=m,ar=kr,ars=krs] endmodule
module sensJ=sensD [ld=lj,d=j,a=g,c=d,h=h,j=m,g=n,b=l,dr=jr,drs=jrs] endmodule
module sensL=sensB [lb=ll,b=l,a=i,d=g,g=j,e=n,br=lr,brs=lrs] endmodule
module sensM=sensB [lb=lm,b=m,a=n,d=j,g=h,e=k,br=mr,brs=mrs] endmodule
module sensN=sensA [la=ln,a=n,b=l,d=j,c=m,ar=nr,ars=nrs] endmodule

rewards "power"
     [ar] true :engr;
     [ars] true:engrs;
     [br] true :engr;
     [brs] true:engrs;
     [cr] true :engr;
     [crs] true:engrs;
     [dr] true :engr;
     [drs] true:engrs;
     [er] true :engr;
     [ers] true:engrs;
     [fr] true :engr;
     [frs] true:engrs;
     [gr] true :engr;
     [grs] true:engrs;
     [hr] true :engr;
     [hrs] true:engrs;
     [ir] true :engr;
     [irs] true:engrs;
     [jr] true :engr;
     [jrs] true:engrs;
     [kr] true :engr;
     [krs] true:engrs;
     [lr] true :engr;
     [lrs] true:engrs;
     [mr] true :engr;
     [mrs] true:engrs;
     [nr] true :engr;
     [nrs] true:engrs;
endrewards
```

APPENDIX C

The p_f Formula

C.1 Notes on Random Sets

Let $V = \{1, 2, \ldots, \ell\}$ be a finite set of elements. $\mathcal{P}(V)$ is used to denote the set of all subsets of V. A *k-subset* $\mathcal{S} \subseteq V$ represents a subset of V containing exactly k elements. Analog $\mathcal{P}_k(V)$ denotes the set of all k-subsets of V. Thus for every k-set $\mathcal{S} \in \mathcal{P}_k(V)$ the uniform discrete probability distribution is given by

$$P: \quad \mathcal{P}_k(V) \to [0,1], \quad P(\mathcal{S}) = \binom{\ell}{k}^{-1} = \frac{k! \cdot (\ell - k)!}{\ell!}$$

In addition the same letter P is used to denote the *uniform probability measure* on the set of all subsets of $\mathcal{P}_k(V)$ with

$$P: \quad \mathcal{P}(\mathcal{P}_k(V)) \to [0,1], \quad P(\mathcal{S}) = s \cdot \binom{\ell}{k}^{-1}$$

where s is the number of elements in $\mathcal{S}$.

Lemma 8 (Random k-Set) *Let e be an arbitrary element from V. The probability that a random k-set contains element e is $\frac{k}{\ell}$*

Proof: The desired probability measure can be computed out of $\frac{b}{a}$ where a is the number of all k-sets and b is the number of all k-sets containing element e. Now, $b = \binom{\ell}{k} = \frac{\ell!}{k!(\ell-k)!}$ while a is the number of all $(k-1)$-subsets of $V \setminus \{e\}$, i.e., $a = \binom{\ell-1}{k-1} = \frac{(\ell-1)!}{(k-1)!(\ell-k)!}$. It thus follows that

$$\frac{b}{a} = \frac{(k-1)!(\ell-k)!\,\ell!}{(\ell-1)!\,k!(\ell-k)!} = \frac{k}{\ell}$$

Corollary 9 *Let e be an arbitrary element from V.*
With Lemma 8 it follows the probability that a random k-set does not contain element e is

$$1 - \frac{k}{\ell} = \frac{\ell - k}{k}$$

Lemma 10 (Random Variable) *A* random variable X *is a real-valued function with*

$$X : \mathcal{P}_k(V) \to \mathbb{R}$$

Definition 11 (Expected Value) *For an element e and a random variable X let $X(\mathcal{P}_k(V)) = \{X(s) \mid s \in \mathcal{P}_k(V)\} = \{e_1, \ldots, e_r\}$ be its range. The quantity $\sum_{j=1}^{r} e_j \cdot P(X = e_j)$ is called the* expected value *of X. In symbols*

$$E(X) = \sum_{j=1}^{r} e_j \cdot P(X = e_j)$$

As usual $X = e$ is shorthand for the set $\{s \in \mathcal{P}_k(V) \mid X(i) = e\}$.

Lemma 12 (Boolean Random Variable) *Let $X(\mathcal{P}_k(V)) = \{0, 1\}$ be a Boolean random variable. Then*

$$E(X) = P(X = 1)$$

Proof As a simple example consider for $i \in V$ the Boolean random variable

$$X_i(S) = \begin{cases} 0 & \text{if } i \notin S \\ 1 & \text{if } i \in S \end{cases}$$

Then $E(X_i) = P(X_i = 1) = P(i \in S) = \frac{k}{\ell}$.

Lemma 13 (Linearity of Expectation) *Let X and Y be random variables on the same probability space, $a, b, c \in \mathbb{R}$. Out of the linearity of the expectation is follows that*

$$E(a + b \cdot X + c \cdot Y) = a + b \cdot E(X) + c \cdot E(Y)$$

Proof See any textbook on probability theory for the simple, but non-trivial proof. Instead of the probability space $(\mathcal{P}_k(V), P)$ we are also interested in products of this space, e.g., in $((\mathcal{P}_k(V))^m, P)$, with $P((S_1, \ldots, S_m)) = P(S_1) \cdot \ldots \cdot P(S_m)$. To avoid overloaded notation we continue to use P for this probability distribution.

Lemma 14 *Let X be the random variable on $\mathcal{P}_k(V)^2$ with $X(S_1, S_2) =$ the number of elements in $S_1 \cup S_2$. Then*

$$E(X) = \ell \left(1 - \left(\frac{\ell - k}{\ell}\right)^2\right)$$

Proof For $1 \leq i \leq \ell$ let X_i be the Boolean random variable on $\mathcal{P}_k(V)^2$ defined by

$$X_i(S_1, S_2) = \begin{cases} 1 & \text{if} \quad i \notin S_1 \cup S_2 \\ 0 & \text{otherwise} \end{cases}$$

By Corollary 9 and Lemma 12 we know $E(X_i) = (\frac{\ell-k}{\ell})^2$. Obviously, $X = \ell - \sum_{i=1}^{\ell} X_i$. By Lemma 13 we get

$$\begin{aligned} E(X) &= \ell - \textstyle\sum_{i=1}^{\ell} E(X_i) \\ &= \ell - \textstyle\sum_{i=1}^{\ell} (\frac{\ell-k}{\ell})^2 \\ &= \ell - \ell \cdot (\frac{\ell-k}{\ell})^2 \\ &= \ell(1 - (\frac{\ell-k}{\ell})^2) \end{aligned}$$

Lemma 15 *Let X be the random variable on $\mathcal{P}_k(V)^n$ with $X(S_1, \ldots, S_n)$ = the number of elements in $S_1 \cup \ldots \cup S_n$. Then*

$$E(X) = \ell \left(1 - \left(\frac{\ell - k}{\ell}\right)^n\right)$$

Proof Straightforward generalization of Lemma 14. Define

$$X_i(S_1, \ldots, S_n) = \begin{cases} 1 & \text{if} \quad i \notin S_1 \cup \ldots \cup S_n \\ 0 & \text{otherwise} \end{cases}$$

We know $E(X_i) = (\frac{\ell-k}{\ell})^n$. Again $X = \ell - \sum_{i=1}^{\ell} X_i$ and the rest of the argument follows as in the proof of Lemma 14.

C.2 Correctness of p_f

C.2.1 PCTL Formula

```
P=? [ true U ("p1") & ("p2") & ("p3") & ("p4") & ("deadlock") ]
label "p1" = (a1=3? true: (n1=2? true: (a1=n1? true: false)));
```

C.2.2 Prism Model Description

```
//Proving the Correctness of the AQF Algorithm
probabilistic

//Bits per Node
const int k;
//number of captured nodes
const int n;//=4;

//Node Array - N
global n1 : [0..2] init 2;
global n2 : [0..2] init 2;
global n3 : [0..2] init 2;
global n4 : [0..2] init 2;

//Adversary
global a1 : [0..3] init 2;
global a2 : [0..3] init 2;
global a3 : [0..3] init 2;
global a4 : [0..3] init 2;

//Bits by Adversary
global u1 : [0..1] init 0;
global u2 : [0..1] init 0;
global u3 : [0..1] init 0;
global u4 : [0..1] init 0;

global c:[0..4] init 0;

module RunNode
     i: [-1..k] init 0;

     [] (i<k) & (n1=2) -> 0.5: (n1'=1) & (i'=i+1) + 0.5: (n1'=0) & (i'=i+1);
     [] (i<k) & (n2=2) -> 0.5: (n2'=1) & (i'=i+1) + 0.5: (n2'=0) & (i'=i+1);
     [] (i<k) & (n3=2) -> 0.5: (n3'=1) & (i'=i+1) + 0.5: (n3'=0) & (i'=i+1);
     [] (i<k) & (n4=2) -> 0.5: (n4'=1) & (i'=i+1) + 0.5: (n4'=0) & (i'=i+1);
endmodule

module RunAdversary
     j: [-1..k] init 0;
     h: [-1..n] init 0;

     [] (j=k) & (h<n) -> (j'=0) & (h'=h+1) & (u1'=0) & (u2'=0) & (u3'=0) & (u4'=0);

     [] (j<k) & (u1=0) & (a1=2) & (h<n) -> (a1'=3) & (u1'=1) & (j'=j+1);
     [] (j<k) & (u2=0) & (a2=2) & (h<n) -> (a2'=3) & (u2'=1) & (j'=j+1);
     [] (j<k) & (u3=0) & (a3=2) & (h<n) -> (a3'=3) & (u3'=1) & (j'=j+1);
     [] (j<k) & (u4=0) & (a4=2) & (h<n) -> (a4'=3) & (u4'=1) & (j'=j+1);

     [] (j<k) & (u1=0) & (a1!=2) & (h<n) -> (u1'=1) & (j'=j+1);
     [] (j<k) & (u2=0) & (a2!=2) & (h<n) -> (u2'=1) & (j'=j+1);
     [] (j<k) & (u3=0) & (a3!=2) & (h<n) -> (u3'=1) & (j'=j+1);
     [] (j<k) & (u4=0) & (a4!=2) & (h<n) -> (u4'=1) & (j'=j+1);

     //Guess the remaining keys
     [] (h=n) & (j=0) & (a1=2) -> 0.5: (a1'=0) + 0.5: (a1'=1);
     [] (h=n) & (j=0) & (a2=2) -> 0.5: (a2'=0) + 0.5: (a2'=1);
     [] (h=n) & (j=0) & (a3=2) -> 0.5: (a3'=0) + 0.5: (a3'=1);
     [] (h=n) & (j=0) & (a4=2) -> 0.5: (a4'=0) + 0.5: (a4'=1);
endmodule
```

C.3 Proving Safety of the AQF algorithm

C.3.1 PCTL Formula

```
const int ComKey;
label "CommonKeys" = (n1=b1?b1:0)+(n2=b2?b2:0)+=ComKey;
P=? [ true U ("CommonKeys") & ("deadlock") ]
```

C.3.2 Prism Model Description

```
//Proving the Liveness Property of the sAQF Algorithm
probabilistic

const int k;   //Bits per Node
const int l;   //Authenticator

//Node Array - N
global n1 : [0..1] init 0;
global n2 : [0..1] init 0;
global n3 : [0..1] init 0;
global n4 : [0..1] init 0;
global n5 : [0..1] init 0;
global n6 : [0..1] init 0;
global n7 : [0..1] init 0;
global n8 : [0..1] init 0;
global n9 : [0..1] init 0;
global n10 : [0..1] init 0;

//Basestation - B
global b1 : [0..1] init 0;
global b2 : [0..1] init 0;
global b3 : [0..1] init 0;
global b4 : [0..1] init 0;
global b5 : [0..1] init 0;
global b6 : [0..1] init 0;
global b7 : [0..1] init 0;
global b8 : [0..1] init 0;
global b9 : [0..1] init 0;
global b10 : [0..1] init 0;


module RunNode
     i: [0..k] init 0;

     [] (i<k) & (n1=0) -> (n1'=1) & (i'=i+1);
     [] (i<k) & (n2=0) -> (n2'=1) & (i'=i+1);
     [] (i<k) & (n3=0) -> (n3'=1) & (i'=i+1);
     [] (i<k) & (n4=0) -> (n4'=1) & (i'=i+1);
     [] (i<k) & (n5=0) -> (n5'=1) & (i'=i+1);
     [] (i<k) & (n6=0) -> (n6'=1) & (i'=i+1);
     [] (i<k) & (n7=0) -> (n7'=1) & (i'=i+1);
     [] (i<k) & (n8=0) -> (n8'=1) & (i'=i+1);
     [] (i<k) & (n9=0) -> (n9'=1) & (i'=i+1);
     [] (i<k) & (n10=0) -> (n10'=1) & (i'=i+1);
endmodule

module RunBasestation
     j: [0..l] init 0;

     [] (j<l) & (b1=0) -> (b1'=1) & (j'=j+1);
     [] (j<l) & (b2=0) -> (b2'=1) & (j'=j+1);
     [] (j<l) & (b3=0) -> (b3'=1) & (j'=j+1);
     [] (j<l) & (b4=0) -> (b4'=1) & (j'=j+1);
     [] (j<l) & (b5=0) -> (b5'=1) & (j'=j+1);
     [] (j<l) & (b6=0) -> (b6'=1) & (j'=j+1);
     [] (j<l) & (b7=0) -> (b7'=1) & (j'=j+1);
     [] (j<l) & (b8=0) -> (b8'=1) & (j'=j+1);
     [] (j<l) & (b9=0) -> (b9'=1) & (j'=j+1);
     [] (j<l) & (b10=0) -> (b10'=1) & (j'=j+1);
endmodule
```

C.3.3 Hyper-Geometric Distribution: Implementation

The implementation using equations (5.9), (5.10) and (5.11) is straightforward. Using the technique of dynamic programming, exponential complexity is avoided. However, to circumvent loss of precision we do not use floating point data types to represent intermediate results. Instead the numbers are stored in `BitInteger` and divisions are postponed to the very end. Consider the following alternations of equations (5.9), (5.10) and (5.11), respectively:

$$P(X_j = c)\binom{l}{k}^j = \sum_{a=c-k}^{c} (\binom{l-a}{c-a}\binom{a}{k-(c-a)}) \cdot P(X_{j-1} = a)\binom{l}{k}^{j-1}$$

$$P(Y = b)\binom{l}{k}^{\tilde{n}+1} = \sum_{a=0}^{l} (\binom{l-a}{b}\binom{a}{k-b}) \cdot P(X_{\tilde{n}} = a)\binom{l}{k}^{\tilde{n}}$$

$$p_f = \left(\sum_{b=0}^{k} P(Y = b)\binom{l}{k}^{\tilde{n}+1} \cdot 2^{k-b}\right) \Bigg/ \left(\binom{l}{k}^{\tilde{n}+1} \cdot 2^{k}\right)$$

Obviously, all divisions have been postponed until the last step, thus all intermediate values are integers. These are stored without loss of precision using the JAVA data type `BigInteger`. The division can then be accomplished with arbitrary precision. We set it to maximum to hold by `double`-variables through this is output data type.
On the downside, the use of such long integers adds an additional factor to complexity. In terms of space it is proportional to the length, in terms of time its length is squared. It can nevertheless be proven that complexity stays polynomial, and specifically, time-complexity $\in O((k + \tilde{n}^2)k^3 l \log^2 l)$ and space-complexity $\in O((k + \tilde{n})kl \log l)$.

APPENDIX D

ESAWN models

D.1 Spin Model

This Model realizes a special case of an ESAWN data-transmission tree. In this case the tree is a chain i.e., all nodes except leaves have outgoing degree of 1. The attacker/corrupt node is enabled to send modified (=wrong) data, but it has to send it exactly when and to the same receivers as a normal node would do.

```
#define N 5     /* nr of nodes, N >= 2 */
#define K 2     /* max nr of corrupt nodes, 0< K <= N-2 */
#define Kp1     3     /* = K+1 */


/* communication channels */
typedef chans {
     /* i sends to j p via channel[j].c[i-(j+1)] */
     chan c[Kp1] = [1] of {byte};
}
chans channel[N];

/* system state variables for verification */
show bool CheatingDetected = false;
show bool AnnounceReceived = false;
show bool ReceivedDataCorrect = false;
show byte FakeNodes=0;
show byte FakePacketsSend=0;
show byte sendData, receivedData;

proctype LeafNode (show byte myid){

     byte data, aggData, aggStore[Kp1];
     byte i = 0;
     chan ch;

     /* Channel Setup */
     atomic {
          do
          :: i <= K ->
               ch = channel[myid].c[i];
               xr ch;
               i++;
          :: i >  K -> break;
          od;
          i = 0;
          do
          :: else ->
               ch = channel[myid - (i+1)].c[i];
               xs ch;
               i++;
          :: i>K || i>=myid -> break;
          od;
          }

     /* generate Data "0" to be send */
     data=0;
     sendData = data;
     /* send data */
     i = 0;
     do
          :: else ->
               ch = channel[myid - (i+1)].c[i];
               ch!data;
               i++;
          :: i>K || i>=myid -> break;
     od;
}

proctype RootNode (show byte myid) {

     byte data, aggData, aggStore[Kp1];
     byte i = 0;
     chan ch;

     /* Channel Setup */
     atomic {
          do
          :: i <= K ->
               ch = channel[myid].c[i];
               xr ch;
               i++;
          :: i >  K -> break;
          od;
          i = 0;
          do
          :: else ->
               ch = channel[myid - (i+1)].c[i];
               xs ch;
               i++;
          :: i>K || i>=myid -> break;
          od;
          }

     /* receive, aggregate and check */
     if
          :: K < N-1 - myid - 1 -> i = K;
          :: else -> i = N-1 - myid - 1;
     fi;
     ch = channel[myid].c[i];
     ch?aggStore[i];
     do
     :: else ->
          /* aggregate */
          aggData = aggStore[i];
          i--;
          /* receive from layer i below */
          ch = channel[myid].c[i];
          ch?aggStore[i];
          /* check aggregates */
          if
          ::aggStore[i] != aggData ->
```

```
                CheatingDetected = true;
                goto ende;
            :: else -> skip;
            fi;
        :: i <= 0 -> break;
        od;

        /* root announces received */
        receivedData = aggStore[0];
        ReceivedDataCorrect = sendData == receivedData;
        AnnounceReceived = true;

        ende: skip;
}

proctype CorruptNode (show byte myid){

        byte data, aggData, aggStore[Kp1];
        byte i = 0;
        chan ch;

        /* Channel Setup */
        atomic {
            do
            :: i <= K ->
                ch = channel[myid].c[i];
                xr ch;
                i++;
            :: i >  K -> break;
            od;
            i = 0;
            do
            :: else ->
                ch = channel[myid - (i+1)].c[i];
                xs ch;
                i++;
            :: i>K || i>=myid -> break;
            od;
            }

        /* receive, aggregate and check */
        if
            :: K < N-1 - myid - 1 -> i = K;
            :: else -> i = N-1 - myid - 1;
        fi;
        ch = channel[myid].c[i];
        ch?aggStore[i];
        do
        :: else ->
            /* aggregate */
            aggData = aggStore[i];
            i--;
            /* receive from layer i below */
            ch = channel[myid].c[i];
            ch?aggStore[i];
            /* Corrupt nodes do not check aggregates and discover forgery*/
        :: i <= 0 -> break;
        od;

        /* aggregate */
        aggData = aggStore[i];


        /* corrupt nodes send any data */
            i = 0;
            do
            :: else ->
                /* send */
                ch = channel[myid - (i+1)].c[i];
                /* generate any data */
                if
                /* correct data */
                :: data = 0;
                /* incorrect data */
                :: data = 1; FakePacketsSend++
                fi;
                ch!data;
                i++;
            :: i>K || i>=myid -> break;
            od;

    ende: skip;
}


proctype HonestNode (show byte myid){

    byte data, aggData, aggStore[Kp1];
    byte i = 0;
    chan ch;

    /* Channel Setup */
    atomic {
        do
        :: i <= K ->
            ch = channel[myid].c[i];
            xr ch;
            i++;
        :: i >  K -> break;
        od;
        i = 0;
        do
        :: else ->
            ch = channel[myid - (i+1)].c[i];
            xs ch;
            i++;
        :: i>K || i>=myid -> break;
        od;
        }

    /* receive, aggregate and check */
    if
        :: K < N-1 - myid - 1 -> i = K;
        :: else -> i = N-1 - myid - 1;
    fi;
    ch = channel[myid].c[i];
    ch?aggStore[i];
    do
    :: else ->
        /* aggregate */
        aggData = aggStore[i];
        i--;
        /* receive from layer i below */
        ch = channel[myid].c[i];
        ch?aggStore[i];
        /* check aggregates */
        if
            :: aggStore[i] != aggData ->
                CheatingDetected = true;
                goto ende;
            :: else -> skip;
        fi;
    :: i <= 0 -> break;
    od;

    /* aggregate */
    aggData = aggStore[i];
    /* non-corrupt nodes send the data on */
    /* send aggregated data */
    i = 0;
    do
        :: else ->
            ch = channel[myid - (i+1)].c[i];
            ch!aggData;
            i++;
        :: i>K || i>=myid -> break;
    od;

    ende: skip;
}

/* -------- init process -------- */
init {
    byte nodeNr, corr;

    atomic {
        nodeNr = 0;     /* start nodes */
        corr = 0;
        run RootNode (nodeNr);
        nodeNr++;
        do
        :: nodeNr < N-1 ->
            if
            :: corr < K ->
                run CorruptNode (nodeNr); corr++;
            :: else ->
                run HonestNode(nodeNr);
            fi;
            nodeNr++;
        :: nodeNr >= N-1 ->
            break;
        od;
        run LeafNode (nodeNr);
    }
    skip;
};
```

Index