

DANIEL CÂMARA

**FORMAL VERIFICATION OF COMMUNICATION
PROTOCOLS FOR WIRELESS NETWORKS**

Belo Horizonte
22 de outubro de 2009

DANIEL CÂMARA

ORIENTADOR: ANTONIO ALFREDO FERREIRA LOUREIRO

**FORMAL VERIFICATION OF COMMUNICATION
PROTOCOLS FOR WIRELESS NETWORKS**

Projeto de tese apresentado ao Programa de Pós-Graduação em Computer Science da Federal University of Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Computer Science.

DANIEL CÂMARA

Belo Horizonte

22 de outubro de 2009

DANIEL CÂMARA

ADVISOR: ANTONIO ALFREDO FERREIRA LOUREIRO

**FORMAL VERIFICATION OF COMMUNICATION
PROTOCOLS FOR WIRELESS NETWORKS**

Thesis project presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

DANIEL CÂMARA

Belo Horizonte
October 22, 2009

© 2009, Daniel Câmara.
Todos os direitos reservados

Ficha catalográfica elaborada pela Biblioteca do ICEx – UFMG

Câmara, Daniel

C172f Formal verification of communication protocols for wireless networks / Daniel Câmara— Belo Horizonte, 2009.
x, 122 f. : il. ; 29cm

Tese (doutorado) — Universidade Federal de Minas –
Departamento de Ciência da Computação.

Orientador: Antônio Alfredo Ferreira Loureiro.

1. Computação - Teses. 2. Redes de computadores –
Teses. 3. Sistemas de comunicação sem fio – Teses.
I. Orientador. II. Título.

CDU 519.6*22(043)



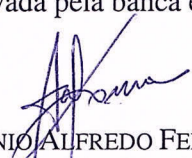
UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

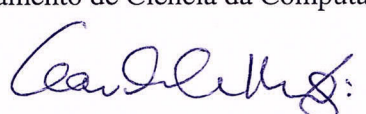
FOLHA DE APROVAÇÃO

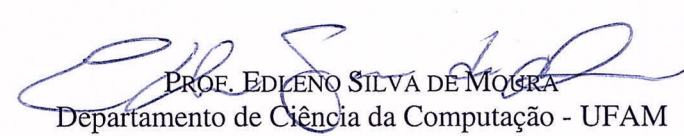
Verificação formal de protocolos para redes sem fio

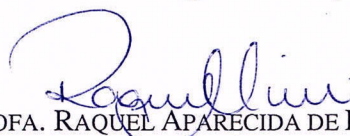
DANIEL CAMARA

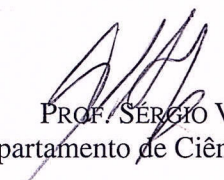
Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:


PROF. ANTONIO ALFREDO FERREIRA LOUREIRO - Orientador
Departamento de Ciência da Computação - UFMG


PROF. CLAUDIONOR JOSÉ NUNES COELHO JÚNIOR
Departamento de Ciência da Computação - UFMG


PROF. EDLENO SILVA DE MOURA
Departamento de Ciência da Computação - UFAM


PROFA. RAQUEL APARECIDA DE FREITAS MINI
Departamento de Ciência da Computação - PUC-MG


PROF. SÉRGIO VALE AGUIAR CAMPOS
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 22 de outubro de 2009.

To my wife, Wanessa, my son, Arthur, and my daughter, Helena.

Acknowledgments

The process I followed to finish the PhD, and the thesis, was not what one can call of standard. For this reason it is amazing the number of people I should acknowledge here, starting by the members of the evaluation committee. Not only for their valuable and insightful comments, but also for the understanding and flexibility you demonstrate for the schedule I had to accomplish. I would also like to thanks to all the secretariat people, mainly Renata Viana Moraes Rocha, Sheila Lúcia dos Santos and Túlia Andrade Salomon Fernandes for the guidance, patience and help over the bureaucratic process. I would like also to thank to Claudemberg Ferreira which whom I work in the beginning of this project, a long, long time ago. Without his work and determination at that time, I am more than sure that nothing would be accomplished here.

I need also to acknowledge and thank FAPEMIG for the scholarship in the beginning of the PhD, that provided me the means to start the whole process. People from Synergia, specially professor Geraldo Robson Mateus, for the confidence he shown and the opportunities he gave me. In special I would like to thank to Eduardo Habib Bechelane Maia, for all the work we developed together and for the support during the time of the thesis defense.

I need also to acknowledge my family for the support and understanding during the time this adventure took to complete. For the lost weekends and for the grumpy temper when the things did not work as expected. Wanessa, thanks for supporting me and providing me the comfort and time I needed to finish this project.

However, above all, I would like to apologize and thank to my advisor Antonio Alfredo Ferreira Loureiro. Apologize for all the trouble and overwork this humble student give to him and thank for the guidance and patience during the whole process. I would like to thank him for convincing me and giving me the opportunity to start the PhD at UFMG, when I was most deluded with my situation, at the time. However, even more, I need to acknowledge him not only for giving me support when I decided to start the second PhD but also to help me finish the first one. Loureiro, I am sincere and deep grateful to you for all you have done to me during all these years.

Resumo

Redes de comunicação sem fio se tornaram nos últimos anos uma constante na vida de um número crescente de pessoas. Em um círculo virtuoso, novos e ainda mais sofisticados protocolos são projetados a cada dia, aumentando o número de ferramentas disponíveis, o que atrai ainda mais usuários. Contudo novas ferramentas são necessárias para ajudar os designers no seu trabalho de desenvolvimento de novos e melhores softwares e protocolos para este tipo de rede. Ferramentas tradicionais, projetadas para ambientes distribuídos tradicionais não necessariamente funcionam no contexto de redes sem fio, devido as características particulares deste meio.

Redes de comunicação sem fio representam um dos piores cenários possíveis para sistemas distribuídos possível. O meio de comunicação não é confiável, mensagens podem ser tanto perdidas quanto corrompidas. Os nodos por sua vez também não são confiáveis, uma vez que dependem de um suprimento limitado de energia que pode se esgotar a qualquer momento. O meio de transmissão é compartilhado e a quantidade de energia gasta com mensagens de controle deve ser reduzido ao máximo, não somente para economizar banda, mas também para salvar a energia dos nodos.

Por exemplo, roteamento é um dos processos mais básicos e importantes em uma rede de computadores. Ter um algoritmo de roteamento correto, robusto e eficiente é fundamental em qualquer rede, e principalmente para redes sem fio. Contudo o principal problema é como garantir estas qualidades desejáveis. Nem simulações nem implementações em testbeds podem garantir a qualidade requerida a estes protocolos.

Simulações são normalmente baratas e criadas executando o algoritmo um número representativo de vezes de forma aleatória. Contudo não há garantias que um cenário específico, que causa um erro, vai estar presente na massa de testes simulada. Testbeds, por sua vez, são significativamente mais caros, e desta forma, tipicamente, com um número menor de nodos. Contudo, ao contrário de simulações, testbeds não sofrem influência das abstrações inseridas nas simulações, uma vez que são implementações reais dos algoritmos. Isto tem seu lado bom, uma vez que problemas reais podem ser detectados, mas da mesma forma que simulações, não existem garantias que os cenários que podem levar a um problema apareceram nos testes realizados no testbed.

Como uma alternativa a estes métodos alguns pesquisadores tem investigado com sucesso o uso de verificação formal como forma de garantir a qualidade dos protocolos de roteamento para redes sem fio. Verificação formal é uma técnica para garantir que um sistema tem, ou não, uma dada propriedade, baseado em uma especificação formal do sistema em avaliação. Esta técnica tem se mostrado uma ferramenta valiosa no desenvolvimento de protocolos para redes sem fio, inclusive contradizendo as afirmações de alguns autores e provas informais.

Esta tese apresenta algumas das principais ferramentas, propostas e técnicas disponíveis para a verificação formal de algoritmos e softwares para redes sem fio ad hoc. A tese também apresenta um novo e simples método para ajudar os desenvolvedores na tarefa de criar protocolos para redes sem fio ad hoc.

Abstract

Wireless networks have become in recent years a constant in the every day life of an increasing amount of people. In a virtuous circle, newer and more sophisticated protocols are designed every day, increasing the available tools and attracting even more users. However newer tools are needed to help designers on their job of creating better software and protocols for this kind of network. Traditional tools, designed for traditional distributed environments, do not necessarily work in the context of wireless networks.

Wireless networks represent the worst distributed system possible. The medium is not reliable, messages can be lost or corrupted. Nodes are also not reliable, once they typically have a limited energy supply, they can fail at any moment. The medium is also shared and the amount of control transmissions must to be reduced to the minimum not only to save bandwidth, but also to save the nodes energy.

For example, routing is one of the most basic and important tasks in a collaborative computer network. Having a correct, robust and efficient routing protocol is fundamental to any network, mainly for a wireless one. However, the problem is how to guarantee these desirable qualities. Neither simulations nor testbed implementations can ensure the quality required for these protocols.

Simulations, normally, are cheap and made by executing the algorithm an expressive amount of times in a random way. However, there is no guarantees one specific case, which may lead to a problem, will be present at the simulated scenarios. Testbeds, on the other hand, are significantly more expensive, and in this way smaller, but, in opposite to simulations, they are not biased by abstractions, they are real implementations. This is good, once real world problems may be detected, but again, there are no guarantees the scenarios that may lead to a problem appear in a testbed implementation.

As an alternative to these methods some researchers have successfully investigate the use of formal verification as a mean to guarantee the quality of routing protocols. Formal verification is a technique that assures a system has, or has not, a given propriety, based on a formal specification of the system under evaluation. This technique has proved to be a valuable tool, even contradicting some authors claims and informal proofs.

This thesis presents some of the main tools, proposals and techniques available to perform formal verification of routing algorithms for wireless ad hoc networks. The thesis also present a new and simple method to help developers in the task of design new protocols for wireless networks.

Contents

1	Introduction	1
2	Background	4
2.1	Formal Verification Techniques	4
2.1.1	Model Checking	5
2.1.2	Theorem Proving	5
2.1.3	Equivalence Checking	6
2.2	Satisfiability and Over vs Under-approximation	6
2.3	The state explosion problem and remedies	8
2.4	Tools	9
2.4.1	HOL	10
2.4.2	SPIN	10
2.4.3	CPN	10
2.4.4	UPPAAL	11
3	Proposals for Verification of Routing Protocols	12
3.1	Formal Analysis of Convergence of Routing Protocols	12
3.2	Formal Verification of Ad-Hoc Routing Protocols Using Spin Model Checker . . .	16
3.2.1	The broadcast system	16
3.2.2	Timers	17
3.2.3	Mobility	17
3.3	A Pragmatic Approach to Model Checking Real Code	18
3.4	A Methodology for Model-checking Ad-hoc Networks	19
3.5	Modeling and Simulation of Routing Protocol for Mobile Ad Hoc Networks Using Petri Nets	20
3.6	An Abstract Model of Routing in Mobile Ad Hoc Networks	21
3.7	Provable Security of On-Demand Distance Vector Routing in Wireless Ad Hoc Networks	22
3.8	Ad Hoc Routing Protocol Validation	23
3.9	Counter-Example Based Predicate Discovery in Predicate Abstraction. Formal Methods in Computer-Aided Design	24
3.10	A Timing Analysis of AODV	26
3.11	Topology Dissemination Based on Reverse-Path Forwarding (TBRPF): Correct- ness and Simulation Evaluation	27

3.12	Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad Hoc Networks	27
4	Methodology	30
4.1	Limitations	31
4.2	Ground Principles	31
4.2.1	Topology abstraction	31
4.2.2	Node position	32
4.2.3	Lower layers services	32
4.3	Modeling	33
4.3.1	Communicating channel	33
4.3.2	Flooding representation	33
4.3.3	Mobility	34
4.3.4	The network	34
4.3.5	Internal and external behavior	34
4.3.6	Information modeling	35
4.3.7	Procedures Abstraction	35
4.3.8	Model	36
4.3.9	Analysis	36
4.4	Algorithm	36
5	Case Study	39
5.1	Modeling	39
5.1.1	Understanding the protocol	39
5.1.2	OLSR State machine	40
5.1.3	Messages and kinds of nodes	40
5.1.4	Dividing into internal and external behaviors	41
5.1.5	Modeling the channel	41
5.1.6	Creating the model	42
5.2	Verifying the model	42
5.2.1	Case study results	43
6	Application to LAR and DREAM Protocols	45
6.1	Evaluated Routing Protocols	45
6.2	Modeling	46
6.3	Results	46
7	Virtual Access Points For Vanets	49
7.1	Related Work	50
7.2	The protocol	51
7.3	Protocol analysis	52
7.3.1	Methodology Application	52
7.3.2	Virtual Access Points for Stream Based Traffic Dissemination	53
7.3.3	Methodology Application	55

7.4	Remarks	55
8	Conclusions	56
8.1	Directions for Future Research	56
8.2	Data Types	111
8.3	Array Variables	112
8.4	Process Types	112
8.5	Atomic Sequences	113
8.6	Message Passing	113
8.7	The statement	114
8.8	Control Flow	114
	8.8.1 Case Selection	114
	8.8.2 Repetition	114
	Bibliography	116

List of Figures

2.1	The Model checking approach	6
2.2	Relation between over and under approximation.	7
2.3	Relationships between the proposals and the tools they use	9
3.1	General View of the relation graphics	12
3.2	Relationships between the proposals and the properties they mainly focus	13
3.3	Relationships between the proposals and the protocols each proposal validate as example of its application	14
3.4	Relationships between the proposals, protocols and tools	15
3.5	Network with three nodes used to evaluate AODV	16
3.6	Hierarchy of pages of the AODV model	20
3.7	General CPN model of a MANET proposed by Yuan et al. [13]	21
3.8	Predicate Abstraction	25
3.9	Chiyangwa Linear Topology Model	26
3.10	Kristensen and Jensen model of network architecture verified	28
3.11	High view of the hierarchy page of the CPN model of Kristensen and Jensen work	28
5.1	Simplified OLSR intermediate node state machine diagram	40
6.1	Delivery failure in LAR1 when a path it is available, problem detected by the methodology	47
6.2	Delivery failure in LAR2, when a path it is available, problem detected by the methodology	47
6.3	Loop detected in LAR2, when a path it is available, and all nodes are in the δ region in a near circle	48
6.4	Loop and delivery failure discovered in DREAM when a series of nodes are in the angle of dissemination is big enough	48
7.1	A road coverage vision	51
7.2	Typical receiving messages map for a 50 APs city scenario, we can see how VAPs allow us to connect existing "connectivity isles"	54
7.3	Typical receiving messages map for a 5 APs road scenario	55

List of Tables

5.1	OLSR Messages and their semantics for each node	41
5.2	Sizes of the OLSR built models	44
6.1	Sizes of the built models for the different algorithms	48
8.1	Data Types	112

Chapter 1

Introduction

This thesis describes the application of formal verification in the development of algorithms for wireless ad hoc networks. It presents a description of the target problem, a review of some of the most important proposals on this field and the results of our current research on this field as well as the next steps and paths we intend to follow in the research.

Developing new protocols and applications for wireless networks is a challenging and error prone task. The transmission medium is not trustworthy, the network topology may be highly dynamic, nodes have limited amount of energy and may fail without any warning. In other words, one of the worst possible distributed scenarios. On the top of that the nodes also share the same medium, so the messages exchange must to be efficient and secure. The exchange of useless messages is not only a waste of bandwidth, but also a misuse of the limited amount of energy. These losses affect not only the sender, but as the medium is intrinsically a broadcast one, any waist with transmissions may be also shared by all nodes in the neighborhood.

Given this scenario and evaluating the number of variables involved in the development of algorithms for this kind of network (e.g. number of nodes, mobility), it is hard to confirm whether a given algorithm is correct or not. For this reason some researchers [1, 2, 3, 19] advocate the use of formal methods as a valuable tool in the development of new algorithms for wireless networks. Our research also indicates that the use of formal methods is a helpful technique to validate the behavior of algorithms for wireless networks.

Formal verification is also starting to be recognized by the computer science community as an important method to validate algorithms. Thus, the winners of the 2007 Turing award [76], probably the most prestigious award of the Computer Science Community, where Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis, “For their role in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries”. This award is the recognition from the community for their efforts and, in some extension, the recognition of the importance of the formal verification technique.

However, at this point two important questions should be answered first, “What is a formal verification technique” and second, “Why should it be applied to routing protocols for wireless ad hoc networks?” In short, the term formal method refers to mathematical-based techniques used in specification, development and verification of software and hardware systems. The use of formal methods intends to increase the rigor on the design and development of systems, leading to more reliable products.

Looking at this definition, and mainly keeping in mind the mathematics involved in the process, some people tend to believe that the use of formal methods, and mainly formal verification, is hard and worthy only for safety-critical systems. However, the fact is that formal methods may help the development of any system and the mathematics involved is straightforward [11]. Formal methods, especially formal verification, can help the protocol designers to decrease the development time [11], find design errors and validate the proposed solutions. Thus the use of such methods tends to improve the final quality of the verified pieces of software. Following this line, this proposal focuses on formal verification as a tool to increase the quality of routing algorithms for wireless networks. Formal verification is the mathematical proof that the formal specified system, and hopefully the developed system, has, or has not, a given property. Such verification can be done manually or automatically.

Normally designers perform a manual verification of a system when they want to understand better the system they are developing. Such proofs aim human readability and, some times, lack the required precision and formalism. Usually manual proofs are done in high level and, not rarely, in natural language. Unfortunately the ambiguity, inherent to the natural language, may lead to subtle errors that can be neglected. Another point to observe is that the continuous improvement in computing capacity has increased the complexity of hardware and software systems. Given such scenario, it is virtually impossible for humans to manually check all aspects of the system. Automatic verification, on the other hand, presents a more accurate method to check the correctness of a system. The use of verification tools also requires a simpler, and more common, mathematical background than the one required to perform a manual verification. This makes this technique accessible to a wider audience and applicable to a broader range of cases.

The current version of this work presents the following contributions:

- A survey of the work on the field, presenting a critical analysis pointing their stronger and weaker points;
- Formal verification and problem identification of four well-known and established protocols;
- Development of a series of new techniques to enable the application of formal verification to a broader range of algorithms for wireless ad hoc networks. Among the problems that could not be formally verified prior than this thesis we can highlight:
 - Topology independence, the presented method does not rely in a specific topology or configuration what makes it more general
 - Number of nodes independence, the technique is independent of the number of nodes in the topology;
 - Flooding, the technique enables, in an efficient way, the verification of flooding based algorithms;
 - Node mobility, the technique permits the verification of the topology dynamic behavior;
 - Specific methodology, the technique presented here is independent and generic. It does not rely in specific characteristics of protocols or tools;

- Development of a methodology that enables the utilization of the described techniques, in a simple way, to a broad range of protocols than the previous techniques;
- Possibility of verification of generic algorithms for wireless networks. The previous works on this field focus specifically on wireless routing algorithms. Our technique, on the other hand, proved useful in the verification of other kinds of algorithms besides routing protocols;
- Creation of a library of verified proceedings to simplify the development and verification of newer protocols for wireless networks.

The remainder of this thesis is organized as follows: the next chapter 2, Background, discusses the main formal verification techniques, main variants, problems and formal verification tools. After that, chapter 3, presents a survey of works of formal verification for wireless networks. Chapter 4, methodology, presents the proposed technique explaining the ground principles it is based on and how to build useful models for communication algorithms. Chapter 5, case study, presents in details the application of the protocol to the OLSR protocol. Chapter 6 presents the results of the methodology application to LAR 1, LAR 2 and DREAM. Chapter 7 presents the Virtual Access Points proposed protocol that was also verified with our technique. Finally, chapter 8 presents the conclusions and some thoughts about future research on the field.

Chapter 2

Background

Formal verification is the process of verifying, based on a series of formal proofs, if a system has or has not a given property. The US Department of Defense DOD 5200.28-STD standard [21], the orange book, states that “a formal proof is a complete and convincing mathematical argument, presenting the full logical justification for each proof step, for the truth of a theorem, or set of theorems, composed as a series of inference steps. This process is machine checkable and each step follows the results of one or more previous steps”.

It is important to point out that formal verification is not a substitute for testing or simulation. These three quality assurance techniques are complementary rather than competitive approaches. They should be used together to improve the system reliability once each one has a different approach and objective.

Test is a way to think how the system works trying to find situations where it may fail. Simulation offers the possibility to run a large battery of tests under identical circumstances where a given parameter can be varied and its effect studied [12]. Formal verification is used to prove the correctness of the system, according to some properties. However, even the most enthusiastic supporters of formal methods recognize that other approaches are important as well [20]. Notice also that neither formal verification nor testing can guarantee that the system is perfect [11].

Dijkstra’s quote about testing, “Program testing can best show the presence of errors but never their absence”, can also be applied to formal verification in the way that it can only prove that a system presents, or not, a characteristic we can think of. However this does not guarantee, by no means, the system is perfect. Even further, the truth is that formal systems are also fallible. The fallibility is the most fundamental limitation of formal verification methods, and it arises from two facts: first, some properties can never be proved and second, we can make mistakes in the proofs of those aspects we want to prove [11].

2.1 Formal Verification Techniques

The most used forms of formal verification techniques in commercial use today [77] are: model checking, theorem proving and equivalence checking. Model checking is a method to verify whether a formally modeled system satisfies a given property [9]. Theorem proving uses mathematical methods, such as axioms or rules, to prove the correctness of a system [10]. Equivalence

checking formally checks if two models, at different abstraction levels, are equivalent [10]. This section discusses these techniques, but it is worth emphasizing that, even though they propose automated solutions, none of them works without some degree of human assistance. For example, sometimes theorem proving requires an advice from the designer of which properties should be verified. Model checkers, on the other hand, can quickly get stuck while checking millions of useless states and human guidance can be handy. Now we will discuss in more detail the three main variants of formal verification techniques: model checking, equivalence checking, and theorem proving.

2.1.1 Model Checking

Model checking verifies if a given model is in accordance with the specification. The model is normally programmed in a special purpose language and it is based on the system specification. Given the complexity of the current systems, the models often represent a simplified version of the target systems. Some tools express the properties to be verified using temporal logic formulae. Temporal logic allows the programmer to express system properties and verify them against the model.

Figure 2.1 depicts the model checking approach. The tool receives as input the system model and the desired/undesired properties to be checked. The output is the answer whether the system holds or not the requested property. In the last case, it usually provides a counter example showing why the property is not satisfied.

In model checking, all the valid inputs and possibilities are verified to guarantee the correctness of the system. To do this model checking tools require a combinatorial amount of states to represent the system. In other words, the number of states required to represent a system increases exponentially with its size, leading to a problem known as state explosion. The state explosion is considered the most important problem in Model Checking [35]. In the last years, many techniques were developed to decrease the effect of the state explosion problem, but it still persists.

The success of the model checking verification depends much on the user's expertise. Because of the state explosion problem, the designer, when building a model, needs to find the right tradeoff between representing the main points of the system and limiting the model size. The model designer must be very careful to do not remove fundamental system characteristics and, at the same time, reduce the system complexity to enable its verification. Building the model and defining the properties to check are critical tasks and must be carefully done. Another problem of model checking is that there is no metric to evaluate the coverage of the verification and, thus, the confidence that the main design properties have been verified.

2.1.2 Theorem Proving

Theorem proving involves the verification of the truth of mathematical theorems postulated about the design. Theorem proving is similar to any other traditional proof: it starts with axioms and using rules of inference the designer tries to prove the truth of a conclusion. The specification of the system is done in first order or higher order logic. From this precise formulation of the system the designer can infer relations to prove its correctness.

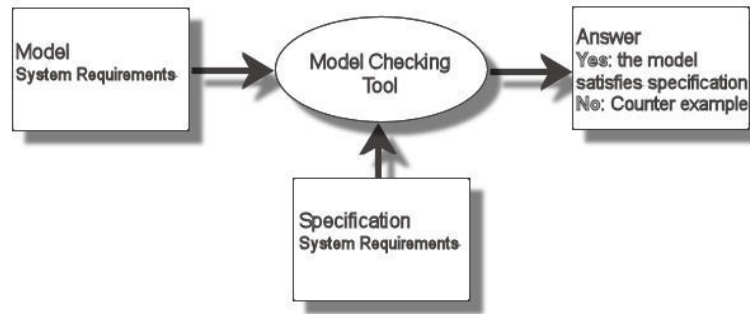


Figure 2.1: The Model checking approach

Theorem proving is probably the most used and that provides the strongest proofs. However, theorem proving tool often requires some guidance from the user and the proof itself can be almost obtained by an interactive process. This technique requires highly trained and experienced designers able to guide the tool through the right path.

2.1.3 Equivalence Checking

Equivalence checking is the process of verifying whether two implementations of the same system, in different abstract levels, are identical. Equivalence checking is very popular in the industry and it is commonly used in the development of digital integrated circuits to formally prove whether two representations of a circuit present exactly the same behavior. In this case, a gate-level implementation is typically compared with its representation at a higher level, the Register Transfer Level (RTL). However, in general, equivalence checking can work well for two structurally similar designs as well.

Notice that equivalence checking does not verify if the design is error free. In addition, when a difference between two design implementations is found, the error diagnosis capability of an equivalence checking tool is, often, limited and, thus, it is difficult to determine the exact cause of the difference.

2.2 Satisfiability and Over vs Under-approximation

A basic component for formal verification tools is a satisfiability (SAT) solver module [84]. This module is the one responsible for deciding if the conditions on a given formula are satisfiable or not. I.e. if the proposed formula evaluates to a true value. The main idea of formal verification techniques is to search in a, possibly, infinite state space if a requested property p is or not present. The SAT module is responsible for determining if p is or not present on such state space. One of the problems with SAT analysis is the size of the considered state space. Abstractions are normally considered an attractive way to overcome this problem [85].

Approximations for solving SAT can either over or under approximate the behaviors of a system. They can be used to guarantee the absence of errors (via over-approximation) and to identify existing errors (via under-approximation). The over-approximation and under-

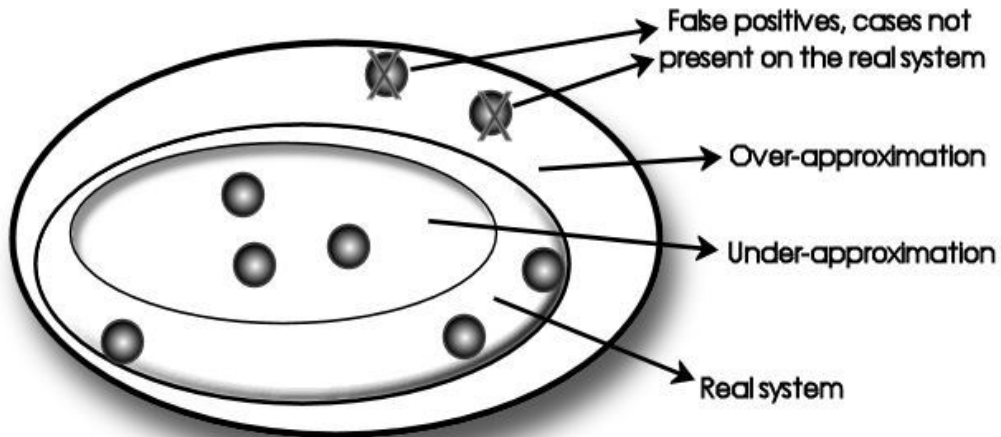


Figure 2.2: Relation between over and under approximation.

approximation differ in the way the real problem is modeled and abstracted.

Under-approximation, of a system S , decreases the scope of S and can be produced, for example, by stopping the construction of the behavior of S at a depth k , thus causing a truncation, $Tk(S)$, of S . In general, for finite state systems, any truncation will be a proper under-approximation of S , in the sense that any computation over $Tk(S)$ is also present in S , but the opposite is not true [86]. Under-approximation presents fewer solutions for the SAT analysis, all satisfying solutions also satisfies p .

Over-approximation of a system, on the other hand, presents all the possible computations of the target system, but possible more. The covering graph $Ck(S)$ generated by the over-approximation of S behavior can map every valid computation of S and every hypergraph reachable from the start graph can be mapped to $Ck(G)$. Therefore, if a given p holds over S , it also holds for $Ck(S)$ [86]. However, not all computations and hypergraphs reachable from $Ck(G)$ may be represented in S , i.e. if p holds for $Ck(G)$ it may, or may not, hold for S . Over-approximation presents a bigger set of solutions for the SAT analysis, a satisfying solutions may, or may not, be valid for S .

The main difference between over-approximation and under-approximation is the size of the solutions set. Over-approximation presents more solutions than the real one, so if the property is unsatisfiable, then so it is p , on the other hand, under-approximation presents fewer solutions, so a satisfying solution also satisfies p . Figure 2.2 presents a graphical representation of these solutions set.

As stated by Breuer and Pickin [87], "In general, exploring more paths than real execution will is preferred, because it leads to flagging false alarms (over-approximation; crying wolf too often) instead of missing alarms (under-approximation; the watchman sleeps on the job)". Our solution focus over-approximation exactly because we also believe it is preferable to have the work to analyze false results than missing a case where a failure could be hidden.

2.3 The state explosion problem and remedies

Model checking tools normally work generating a Binary Decision Diagram (BDD), that is a compact structure to represent all the states for the target system. However, creating the BDD is not enough to formally verify a system. It is needed to compute the system reachable states. Reachability analysis has been proved to be one of the most effective techniques to formally verify a broad range of systems. It consists of the analysis of which states the system can reach in the next steps, giving the current state. Although it is a powerful technique, it has its application severely restricted due to the state “explosion problem” [19]. This term refers to the situation in which the state space storage grows exponentially with the size of the model. The state space explosion problem occurs because of the large number of possible interleaving between processes in a reactive concurrent system. In this case the verification may fail, not because the model is wrong, but simply because there is not enough memory to verify the target system [12].

A number of proposals have been made to minimize this problem, and, thus, enable its application to the verification of real systems. In the following, we list some of the main techniques, according to the description provided by Clarke et al. [25]:

- *Symbolic representation*: This technique refers to the use of compact data structures to represent the state space [26], i.e., encoding the transition relations of a Kripke structure as a Binary Decision Diagram (BDD). In this case it is possible to save storage space by exploiting the often inherent regularity of a hardware or software system. Other example of symbolic representation is the constraint system representation of continuous parameters such as clock ranges, i.e., UPPAAL [39]. In this case it would not even be possible to store explicitly all-time points, regardless of the amount of memory available [12].
- *Partial order reduction*: It is based on the principle [27] that if two, or more, processes do not exchange information during their lifetime, it does not matter if they run in parallel or in any sequential order. This makes the verification easier since these processes can be verified in isolation from each other. Partial order reduction analyzes the processes execution and exploits the commutativity of concurrently executed transitions, which result in the same state when executed in different orders. Notice that the verification property must also be taken into account since it might introduce additional data dependencies between processes. Partial order reduction has been successfully applied to a series of tools, such as SPIN [28].
- *Compositional reasoning*: In short, it is the decomposition of the system into components which are verified apart from the other components [29]. Given the composition of these parts, global properties can then be inferred. Even if mutual dependencies between components exist, the components can be verified separately assuming that the other components work as expected.
- *Abstraction*: Abstraction is a way to decrease the complexity of system models [30]. Normally, when modeling a system, one may use abstractions in many ways. For example, instead of verifying the behavior of the system for all possible floating inputs, different classes of values can be modeled and used. When modeling network protocols, normally,

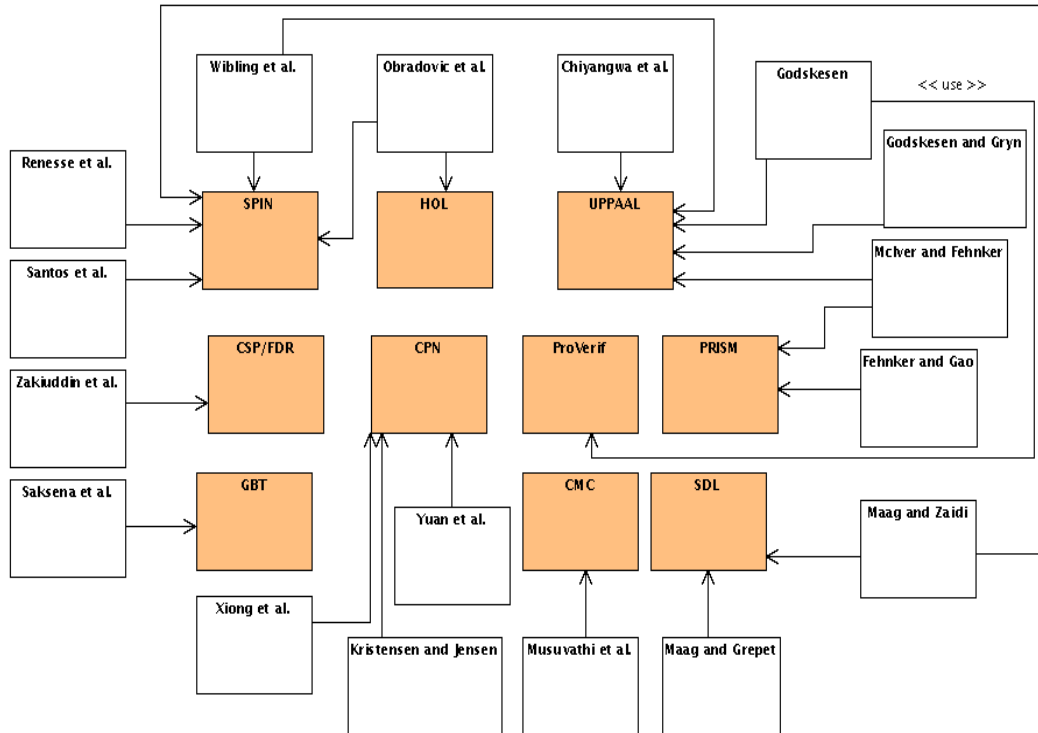


Figure 2.3: Relationships between the proposals and the tools they use

other stack layers and protocols are abstracted from the problem to decrease the complexity of the model. Automatic abstraction methods are also available and can help in the formal verification of a broad range of systems.

- *Symmetry*: Many systems are symmetric in their design and implementation. Some times this symmetry can be seen as a form of redundancy [31]. Symmetry reduction [32] is a technique that combines states, which are similar, into equivalence classes. From these a new reduced model is built choosing one representative of each equivalence class. Hopefully the new model will be smaller than the original one preserving the state transition graph. Using this technique it is possible to reach a substantial, often exponential, savings in terms of states.

2.4 Tools

The automatic verification of protocols is intrinsically linked to software tools. This section briefly presents some tools used in the literature to formally verify communication algorithms for wireless networks. All these are general tools and can be applied to a number of different applications to verify a broad range of systems. Figure 2.3 presents a list of the tools used in the proposals described in Chapter 3 and the protocols verified with them.

2.4.1 HOL

Higher Order Logic (HOL) [36] is a powerful and widely used interactive theorem proving tool. It is used to construct formal specifications and proofs in higher order logic. HOL is used in a broad range of areas and problems, being successfully in both industry and academia. HOL is a complete programming environment in which theorems can be proved and proof tools implemented. An important characteristic of this tool is its high degree of programmability based on the ML meta-language. To help the developers, HOL has some built-in decision procedures and theorem proofs. An oracle mechanism also gives access to external programs such as SAT and BDD engines. Obradovic et al. [2, 14] used HOL to verify the AODV protocol.

2.4.2 SPIN

SPIN is an efficient verification system for modeling distributed software systems. It provides a powerful and concise notation for expressing general correctness requirements [17]. SPIN accepts design specifications written in the verification language PROMELA (Process Meta Language) [18] and the specification or correctness properties are expressed in linear temporal logic (LTL). The description of a concurrent system in PROMELA consists of one or more user defined process templates and at least one process instantiation. The templates define the behavior of different types of processes. Any running process can instantiate further asynchronous processes, using the process templates [17]. SPIN translates each process template into a finite automaton.

Given a model system specified in PROMELA, Spin can either perform random simulations of the system's execution or it can generate a C program that performs an efficient online verification of the system's correctness properties [82]. This saves memory, improves performance and allows the insertion of C code directly into the model. SPIN was used in a number of proposals [1, 4, 14, 22] to verify different routing protocols for wireless ad hoc networks, such as AODV, WARP, LAR, DREAM, LUNAR.

2.4.3 CPN

Coloured Petri Nets: Petri Nets [37] is a tool that allows the creation of mathematical representations of discrete distributed systems in an intuitive and graphical way. The systems are modeled as graphs consisting of place nodes, transition nodes and directed arcs connected by transitions. In Petri Nets, modules interact using a set of well-defined interfaces. The graphical representation makes it easier to see the basic structure of a complex model.

An important characteristic of Petri nets it is that they can handle more than one data stream at each time. This provides great expressiveness especially when modeling distributed and parallel systems. There are two main variants of Petri nets: the standard one and Coloured Petri Net (CPN) [38]. Unlike standard Petri nets, where tokens are indistinguishable, in a Coloured Petri Net, every token has a value. This makes easier for the designer to express different events and actions. Coloured Petri Nets have also a formal, mathematical representation with a well-defined syntax and semantics. Petri nets have been used to prove the correctness of AODV [23], DSDV [13] and ERDP [24].

2.4.4 UPPAAL

UPPAAL [39] is a tool suited for modeling, simulating and verifying a broad range of systems, but mainly real-time systems. To do so it uses a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels and/or shared variables.

UPPAAL has three main parts, a description language, a simulator and a model checker. The description language is non-deterministic and serves to describe the system behavior using a network of timed automata. The simulator is used for interactive and automate analysis of the model, and for the verification of the correctness of the programmed model examining specific executions of the model. The model checker can also be used in an interactive way to find errors in the modeled system. However its full power is shown when automatically covers the exhaustive dynamic behavior of the modeled system. UPPAL automatically generates a diagnostic trace that explains why a property is, or is not, satisfied by a system description.

Chapter 3

Proposals for Verification of Routing Protocols

This chapter presents a survey of related work on the field of formal verification for wireless networks. We also present the relation of the works through a series of graphs. Figure 3.1 presents a the relation among these graphs. Figure 3.2 presents the relationships between the proposals and the properties they verify. Figure 3.3 shows the relation between works and verified protocols and Figure 3.4 shows a more complete view with the relations of proposals, protocols and tools.

3.1 Formal Analysis of Convergence of Routing Protocols

Obradovic et al. [2, 14] use the theorem prover HOL and the model checker SPIN together to prove key properties of distance vector routing protocols. To exemplify their method they use it to verify the Ad-Hoc On-Demand Distance Vector (AODV) protocol [63]. Perkins and Royer [63] affirm that AODV is loop free, they even give a sketch of a proof for such claim.

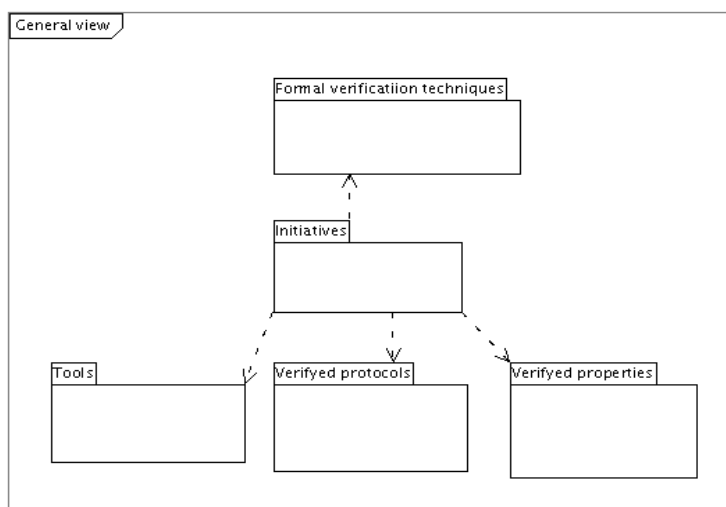


Figure 3.1: General View of the relation graphics

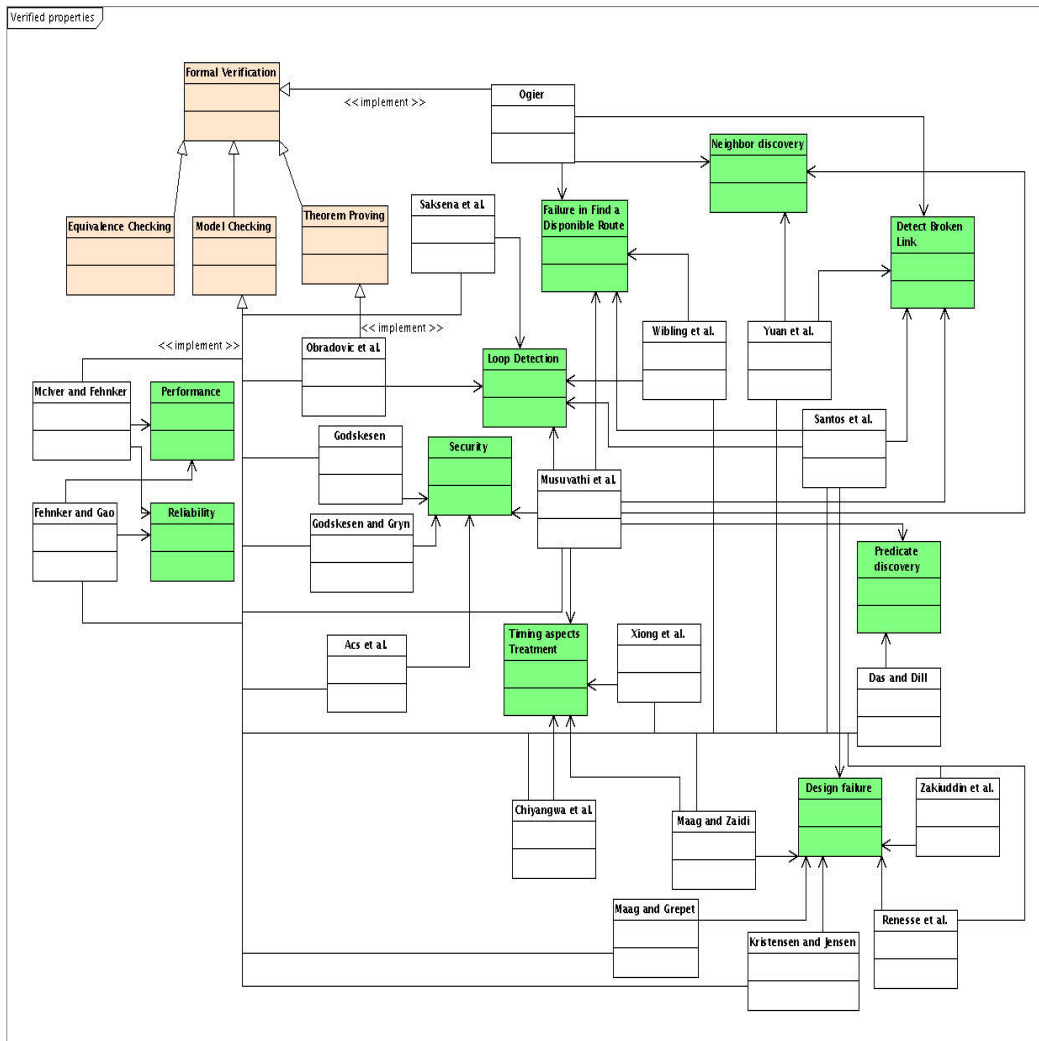


Figure 3.2: Relationships between the proposals and the properties they mainly focus

However Obradovic et al. manage to detect some flaws at the routing loop prevention mechanism of AODV. As a consequence, modifications were made in the new versions of AODV loop prevention mechanism. This fact shows the importance of the use of more formal methods in the development of new routing algorithms.

The reason why Obradovic et al. decided to use both HOL and SPIN it is because the two tools clearly have different payoffs and objectives. SPIN is more suited to model and simulate communication protocols and it has some fixed verification strategies to do it. On the other hand, HOL offers a more powerful mathematical infrastructure, allowing the user to develop more general proofs [2]. As drawbacks, normally SPIN verifications are limited by memory restrictions and expressiveness. HOL verifications, on the other hand, are bound by the complexity and time required to reach a result. The technique proposed by Obradovic et al. consists in coding the protocol first in SPIN and use HOL to address limits in the expressiveness of the SPIN model. They use HOL to prove abstractions addressed in SPIN. For example, if a property P stands for two routers, it will stand for arbitrary routers. In this way HOL reduces the memory required by SPIN while ensure the model implemented in SPIN is correct.

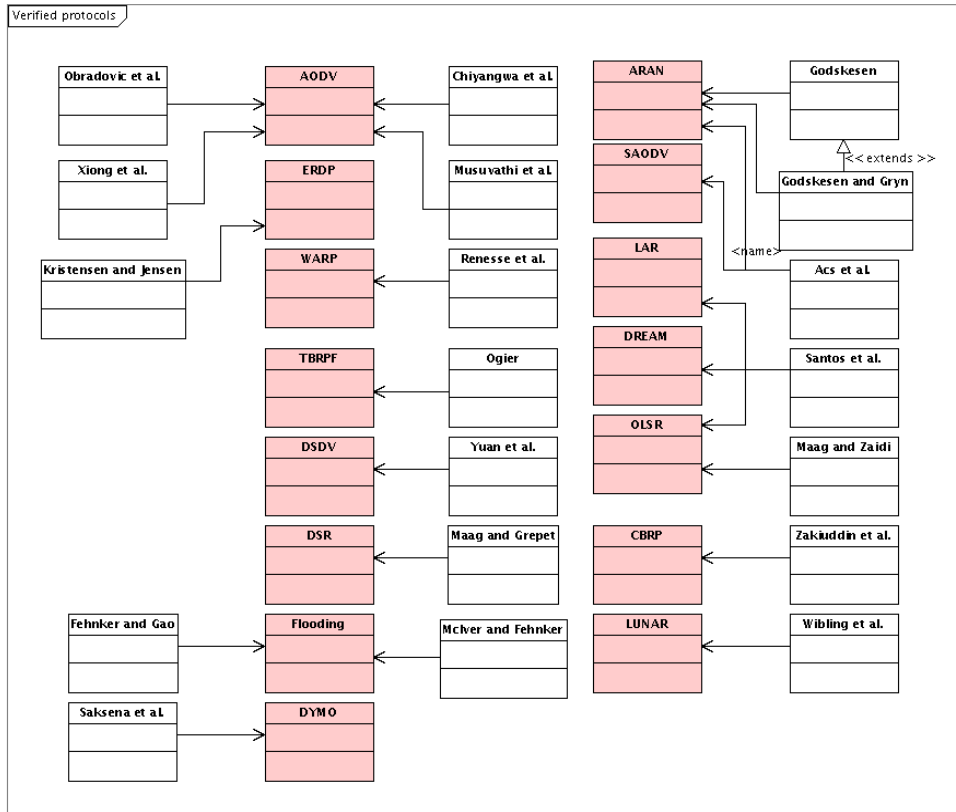


Figure 3.3: Relationships between the proposals and the protocols each proposal validate as example of its application

The verification of AODV was done for the network topology presented in Figure 3.5. The verification of the AODV model considers all three nodes. D is the only destination, and both A and B attempt to send data to D . The link between B and D is fragile and may be broken at any time. A challenge is to discover that the B - D link has broken and the route to D is no longer available. Note that, if A and B form a routing loop, they will never discover that D is unreachable.

During the verification, SPIN found a number of scenarios where the routing loop could occur. After the analysis of such scenarios, it was realized that three of them were routing loop situations.

An important achievement of that work is to identify the omissions and errors at the AODV specification that can lead to loop situations. Guided by the scenario results, Obradovic et al. defined three assumptions that are proposals to change the AODV specification to avoid loop situations:

- $A1$ – Whenever a node discovers that its route to a destination has expired or broken, it increments the sequence number for the route;
- $A2$ – Nodes never delete routes;
- $A3$ – Nodes always immediately detect when a neighbor restarts its AODV process. The restart is treated as if all links to the neighbor have broken.

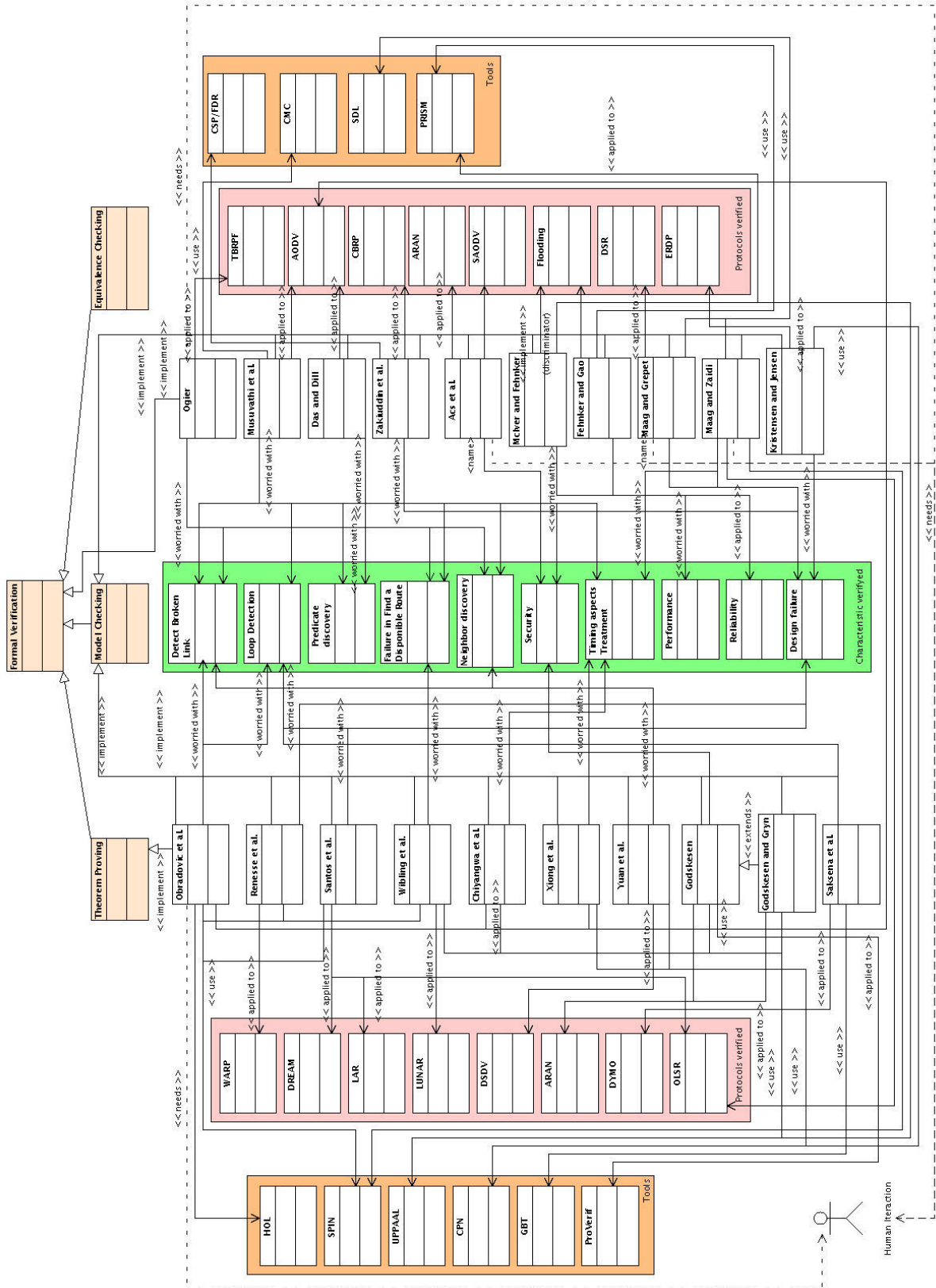


Figure 3.4: Relationships between the proposals, protocols and tools

These modifications lead to **Theorem 1** “Consider an arbitrary network of nodes running

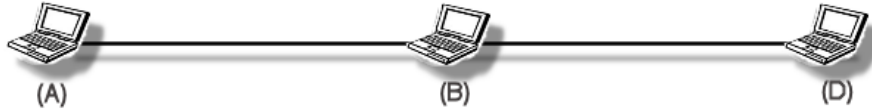


Figure 3.5: Network with three nodes used to evaluate AODV

AODVv2. If all nodes conform to the assumptions A1-A3, there will be no routing loops formed". The complete proof that the new protocol is loop free is presented using HOL, based on a corollary of the preservation of a key path invariant of the protocol, used also to prove route validity [2].

The main problem with this verification approach is the strong dependency on the user [13]. HOL is a semi-automatic theorem prover and the user needs to guide it. Another problem is the complexity in defining the theorems and lemmas to prove. This definition must be done carefully by an expert user.

3.2 Formal Verification of Ad-Hoc Routing Protocols Using Spin Model Checker

Renesse and Aghvami [22] present a technique to use SPIN to formally verify routing algorithms for ad hoc networks. In their work Renesse and Aghvami argue that the supertrace mode, or bitstate hashing [83], a partial checking method, of SPIN is more suitable for large models, as it is the case of routing algorithms for ad hoc networks. With the supertrace mode of SPIN, validations can be performed with less memory, and still retain reasonable coverage.

The bitstate hashing method is a partial check technique, and some states may be missing. To predict the coverage, SPIN uses a hash-factor [64]. The coverage function is defined as $Hf = \frac{M}{N'}$. Where N' is the number of states reached and M the maximum number of storable states. When $Hf > 100$ the coverage is $> 99.9\%$, if Hf is between 10 and 100 the coverage is greater than 98%.

The SPIN tool is normally used to verify communication protocols with a fixed topology. Renesse and Aghvami propose ways to represent broadcast, timers and mobility to verify protocols for mobile networks with SPIN.

3.2.1 The broadcast system

In SPIN, when two nodes want to communicate they use a predetermined channel. To emulate a broadcast, a node must have as many channels as nodes in the network. Each channel is associated with a different node and each node can receive messages only through this channel.

An example of a PROMELA code to perform a broadcast:

```

if
  :: (routing-tab1[k].next-id == 2) ->
      broadcast2!lu,source-id,destination-id
  :: (routing-tab1[k].next-id == 3) ->

```

```

        broadcast3!lu,source-id,destination-id
    :: (routing-tab1[k].next-id == 4) ->
        broadcast4!lu,source-id,destination-id;
fi;

```

3.2.2 Timers

PROMELA does not provide real timers, or at least not in the sense routing protocols need. Renesse and Aghvami argue that timers are just triggers from the point of view of the verification process and could be implemented in this way:

```

if
  :: (timer=1) ->
      goto cancel-entry
  :: (timer=0) ->
      Skip
fi;

```

This means that the entry timer reaches its value or not. In this case SPIN will take care of verifying both possibilities. In a routing table, for example, each entry should have its own timer.

3.2.3 Mobility

A node in an ad hoc network cannot differentiate its mobility from the mobility of other nodes. The node mobility is just perceived as broken links. Renesse and Aghvami propose, without using timers, to simulate mobility reconfiguring the routing table of the nodes. The moving node loses all the entries and the others remove the moving node from their own routing tables.

Another way is to let SPIN compute all different configurations using the case selection feature of PROMELA. In the initialization case, the links setup can be specified in this way:

```

if
  :: (neigh-id=2) ->
      broadcast2!1,my-id,destination-id
  :: (neigh-id=3) ->
      broadcast3!1,my-id,destination-id
  :: (neigh-id=4) ->
      broadcast4!1,my-id,destination-id
fi;

```

Doing the initialization as stated, all initial configurations will be verified once SPIN checks every possibility at the case selection. This second approach is easier to implement and to check using the tool.

To validate their model, Renesse and Aghvami verify the Wireless Adaptive Routing Protocol (WARP) [65]. WARP is a hybrid table-driven on-demand protocol. WARP attempts to

maintain up-to-date routes between all nodes in the network with routing tables and link-update propagations. If there is no route in the routing table to the destination WARP uses a routing discovery process.

To verify WARP, the authors use a network with five nodes. They argue that it is enough to provide indication of the behavior for larger networks. Each node has a routing table with the following entries: destination, next hop, backup next hop, and adjacent nodes timer. They also model hello and link update messages sent among the nodes. The most critical part of the method is the complexity of the algorithm reduction. The simplified model of WARP has one fifth the size of the complete model.

The main simplifications of the model are related to the number of nodes, number of links, number of sent messages and order of arrival of the nodes. The five nodes in the network are: one source, one destination and three intermediate nodes. Each node can initialize only one link with another node. In the model, node one sends only one message to node five and cannot be directed connected to it. All nodes arrive at the network one after the other. The simplifications, even reasonable, are very restrictive and a series of possibilities are not verified. This may compromise even the validity of the results. Another point to observe is that the authors simply state that five nodes are enough, but there is no analysis or strong justification to use this number.

3.3 A Pragmatic Approach to Model Checking Real Code

The process of building models for a model checker tool is a hard and error prone task. When an abstraction of the code, rather than the code itself, is verified it is easy to miss possible errors. Observing this Musuvathi et al. [40] suggest a new way to perform formal verification. They propose a new model checker, CMC (C Model Checker), which checks C and C++ code implementations directly, eliminating the need for a separate abstract description of the system behavior. Performing the verification on the real code the designer neither misses the errors that would be omitted from a model nor wastes time evaluating bugs that appear in the model but not in the real implementation.

To validate CMC, the authors applied it in the verification of an implementation of Ad-hoc On-demand Distance Vector (AODV) [63]. They found 34 errors in the code including one error in the AODV specification itself.

Before starting the verification using CMC, it is necessary to specify the correctness properties, i.e., the invariants that should hold for all cases. Some properties are domain independent, as the case that the program should not access illegal memory area. Other properties are domain dependent and must be placed at the code through assertives. The second step to apply CMC is the definition of the environment. The user must build a test environment that adequately represents the behavior of the actual environment in which the protocol is executed. The third and last step is the definition of the initialization functions and event handlers for every process in the system.

CMC is a very interesting and promising tool, however, when CMC compromises with the analysis of a real C/C++ implementation, it verifies that version of the protocol that was

implemented, it became harder to verify if the protocol specification has design errors. What the tool is doing in the end is just evaluating errors on that specific implementation of the protocol. Even though the authors argue in contrary [40].

For example, Obradovic et al. [2] found a number of flaws at AODV specification exactly because they were not bounded by a real implementation. Specifically for routing protocols, one should additionally consider that the method ignores the interaction among nodes ,in other words, how to verify the dynamic behavior of the protocols. Possibly, one could create a program that models this interaction and afterwards verify this program. This could be possibly performed through the identification of initialization function and environment specification phases. However, it would be harder than simply modeling the desired protocol behavior and verifying it. More over Wibling et al. [69] argue that CMC approach is not aimed at proving correctness, but rather it is a tool to find bugs in the code since an exhaustive state space search can generally not be performed.

3.4 A Methodology for Model-checking Ad-hoc Networks

Zakiuddin et al. [41] propose a methodology to verify ad hoc network protocols through model checking. Their approach is limited to a small number of nodes, typically about five. The authors argue that this is enough to characterize undesirable behavior. They also argue that given the characteristics of the data and the tool they use, *CSP* and *FDR*, the results are applicable to an unbound number of nodes, but do not present a proof of that. They study a military network and so, for confidentiality reasons, they do not give many data about the network itself.

The first approach they use to verify the protocol is the simple translation of the protocol into *CSP_M*. The only simplification made was in the range of the continuous parameters and the number of data types. As expected this approach leads to a complex and intractable model.

From that the authors tried a different approach, starting from the simplest possible representation of the system. With this model they simply capture the states of the system components and then map interactions between such components onto their states. Step by step, the complexity of the model is increased while most of the network behavior is added.

The technique relies on two characteristics of the tool *CSP_M*. The support for renaming a process and its model of shared event communication. The renaming is used to program the nodes interaction, and the model of shared event communication is used to implement this interactions.

The methodology proposed by Zakiuddin et al. consists of essentially of three parts [41]:

1. **The logical view:** The *CSP_M* processes should be modeled to capture the state transitions of the units of the verified protocol. The model must include processes for links as well as nodes. These basic processes will only capture local states and state transitions;
2. **Promoting local to global:** *CSP_M* renames the map interactions onto the states of the basic processes. Indeed, the local states of the node and link processes can affect each other, allowing in this way both local and system views;
3. **Specifying properties:** Code the requisite properties of eventual correctness in *CSP*.

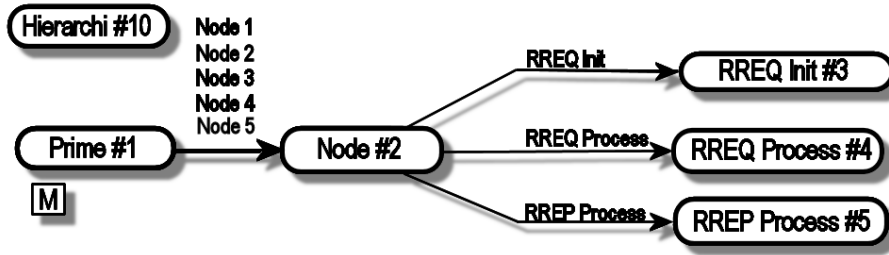


Figure 3.6: Hierarchy of pages of the AODV model

Authors emphasize that an important part of their technique it is to condense as much the protocol as possible in one single shared event while the local view process stays as simple as possible. Zakiuddin et al. [41] apply the technique to the Cluster Based Routing Protocol (CBRP) [66] based on the following assumptions:

1. All links are bidirectional;
2. Routing tables and routing table information are not stored nor transmitted. The needed information is got implicitly;
3. Nodes always receive messages that change their behavior.

Although the authors claim about the specification of a methodology, the proposed technique relies heavily on specific characteristics of the used tool. A point to notice is that the application of the methodology depends on the user proficiency of the tool. Once the procedure to apply the methodology is not fully specified, two people applying the same technique may arrive at different implementations and possibly results. Another point to observe is the assumptions with respect to modeling. Some of them may be appropriate for CBRP, but can be very restrictive to others, and there are no guidelines for choosing such assumptions nor for ignoring some of them, if needed.

3.5 Modeling and Simulation of Routing Protocol for Mobile Ad Hoc Networks Using Petri Nets

Xiong et al. [23] propose a timed model for AODV protocol based on the idea of topology approximation mechanism, which describes the aggregate behavior of nodes when their long term average behaviors are of interest. With this technique the nodes and their relationships are modeled as a graph where nodes are the vertices and the links the edges of the graph. With this the vertex degree shows the number neighbors of the node. This structure is then translated into Colored Petri Nets (CPN). Figure 3.6 shows the model hierarchy of pages. The top level of the model is called prime, the second level has the template of the node that implements AODV and on the third level has the pages that represent the AODV states.

This work introduces a new approach to verify routing protocols for wireless ad hoc networks and present a very reliable architecture to do so. However it also makes a series of strong assumptions about the network:

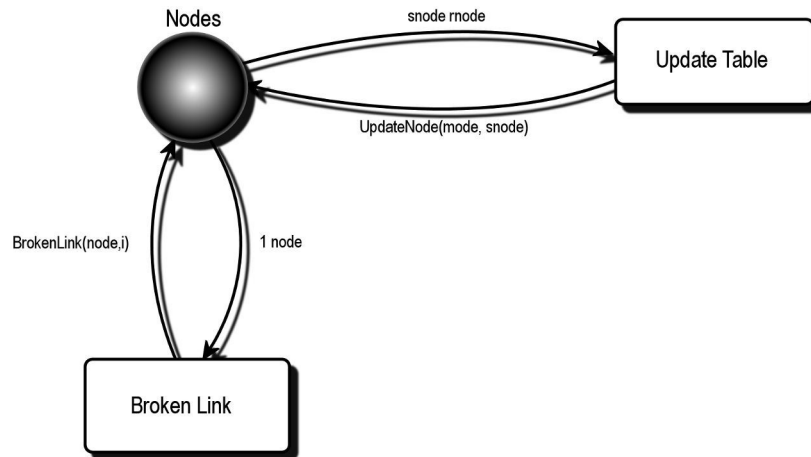


Figure 3.7: General CPN model of a MANET proposed by Yuan et al. [13]

- Links are symmetric;
- Every node knows all their neighbors;
- All nodes have the same transmission range.

Besides these assumptions the authors do not have any study about the number of nodes, ideal neighborhood or even how topology changes affect their AODV model. It is only said that it is possible, in their architecture, to instantiate as many nodes as possible, but what exactly this means is not clear. To perform the verification that work uses five nodes, but there is no explanation about why to use such number. The protocol verification is, partially, bounded by the computational power available to the user, i.e. if there are more resources you can add more nodes. We believe that this characteristic should be avoided in formal verification. The verification model of a protocol should be resource independent. If one proves the validity of a protocol for a given configuration, this proof should stand for other configurations as well. After all, what it is important is to know whether the protocol is correct or not, regardless of the amount of resources available for the verification process.

3.6 An Abstract Model of Routing in Mobile Ad Hoc Networks

Yuan et al. [13] illustrate the dynamic operations of a MANET using Coloured Petri Nets. They show a simple, yet powerful, way to model the dynamic topology changes of ad hoc networks with Colored Petri Nets. This approach is simpler and presents less unrealistic assumptions than the one presented in [23] that also uses CPNs.

The model expresses how nodes update their routing tables and how they deal with broken links. The authors verify the DSDV protocol to exemplify their technique, Figure 3.7 presents the basic model based on the DSDV protocol.

In this verification approach, each node begins with only one entry in its routing table pointing only to itself. The Update Table and Broken Link events are independent and can occur at any time. The Update Table receives an acknowledgment message with two variables – s_{node} and r_{node} – that represent two arbitrary network nodes. The update can occur at any time. The function *UpdateNode* returns the changes of the routing table of the r_{node} if it is in the range of s_{node} in the moment that the update message was sent.

In Figure 3.7, the transaction *Broken Link* models a node that detects it has not received an update from a neighbor in the expected period of time. This event occurs in a non-deterministic way and when it occurs an arbitrary $node(i)$ is pointed as the one that lost the connection. The chosen node is always a neighbor, or in other words it has a *hopcount* of one from the actual node, guaranteed by the $neighbor(node, i)$ function in the model. The function *BrokenLink*($node, i$) updates the routing table of the node with the new information and the function *UpdateNode*($node, snode$) returns the updated node routing table.

Yuan et al. [13] design for simplicity and elegance of the proposed model to handle mobility. However, regarding the model simplifications, the nodes do not really send messages, so there are no differences between full and incremental routing table updates. This simplification may hide important errors that are not verified. It is also important to notice that, as consequence of how the transmitted messages are modeled, two different nodes cannot receive and process broadcast messages at the same time. In this case, errors caused by concurrent sending/receiving messages cannot be verified.

Even though the model can represent the dynamic behavior of a MANET, it does it in an arbitrary way. The technique depends on node configurations created arbitrarily, i.e., there is no guarantee that a configuration that causes a problem is represented in the proof.

3.7 Provable Security of On-Demand Distance Vector Routing in Wireless Ad Hoc Networks

Ács et al. [42] proposed a framework model to verify security of on demand routing algorithms. They apply their model to Secure AODV (SAODV) [67] and Authenticated Routing for Ad-Hoc Networks (ARAN) [68] protocols. Basically they propose the creation of two distinct models, a real world and an ideal world model. The real world model should describe the real operations of the system, and the ideal world model should capture what the system wants to achieve in terms of security. To prove the system security, the outputs of these two models must be indistinguishable [42]. The ideal world model is secure by construction, since no attack can be successful. On the other hand, attacks can be successful at the real world model, once no precautions are made in the sense to avoid such attacks.

Ács et al. model the network as a graph $G(V, E)$, where V is the set of vertices and E is the set of edges. The vertices represent the nodes and the edges the radio links among them. To simplify the model, the authors assume that links are symmetric.

Besides the theoretical justification to this work, an important point is the new standpoint to the problem. The authors argue that if a model is specified in such a way that every one of its aspects works perfectly, to prove that the other system is correct it is enough to compare the

output of both systems. Of course it might be very difficult to define the ideal world model, and which characteristics must be modeled to achieve the objectives of the correctness of the system. However, in some cases it is worth to think in this way and represent the system according to these principles.

On the other hand, the proposal has some drawbacks. First of all, the process is still theoretical and no automated proof is presented. Furthermore, it is not clear how the results were obtained nor what it is the real scope of the model. Regarding the implementation, the model is set to run on a “perfect” architecture composed of Turing machines interconnected by the same tape. This is convenient, as a theoretical model, but the real implementation, to be useful, should be very different from that.

Ács et al. emphasize they verify only the security of the route discovery part of the routing protocol. They are not concerned about a node misbehave as in the case that a node intentionally does not forward a message. In this way the behavior of protocols under attacks, such as worm holes and deny of service, is not verified.

The occurrence of attacks depends on a specific network configuration. Thus, the modeled attacks should be planned very careful to work, even in the real world model. For example, if an attack depends on a particular five-node configuration to work and there are only four nodes, or five in a different configuration, this attack may not even work in the real world model.

3.8 Ad Hoc Routing Protocol Validation

Wibling et al. [69] present one of the most complete and interesting work on this field. They develop a series of techniques applied in the verification of the Lightweight Underlay Network Ad-hoc Routing (LUNAR) Protocol [70]. They use both SPIN and UPPAAL. With SPIN, the data and control aspects of the protocol are verified. UPPAAL serves in the verification of timing properties. In both cases the size of network, i.e., the number of nodes involved, as well as the nature of the topological scenarios are limited due to state space storage overhead [69].

Besides the proposed verification techniques, that work also defines what it means a “correct operation” for an ad hoc routing protocol: “If there at one point in time exists a path between two nodes, then the protocol must be able to find some path between the nodes. When a path has been found, and for the time it stays valid, it shall be possible to send packets along the path from the source node to the destination node” [69]. This definition is important because it expresses how a generic protocol should work, independently of its internals. Another important fact about this definition is that it is also easily expressed in terms of mathematical logic. This definition has also been used by Chiyangwa and Kwiatkowska [3] in their work.

Wibling et al. build a formal model, based on model checking, that is an abstraction of the real protocol. To construct such model one must evaluate the tradeoff between the verification complexity and relevance of the results [12]. In this way the formal system model can be checked to fully comply with a given set of requirements. The work proposes a model of layers above and below the network layer, responsible for routing process. Upper layers are seen as generators and destinations of IP packets destined for a certain network address. For the link and physical layers, the authors specify if nodes are connected or not, and assume that no intermediate state

is possible. Connected nodes can send packets to each other without loss or corruption during the time of connectivity. Using UPPAAL, for verification of timing properties, they additionally include a nondeterministic packet delivery delay [12].

They propose to model each protocol instance and the connectivity between the nodes explicitly. If at any time the network topology changes, such change should appear in the model. This approach limits the number of nodes and types of possible transitions to be verified. Indeed, the maximum topology size the work can handle is a six-node topology where any node can become unavailable at any time with the properties being verified automatically, without human interference.

To deal with the state explosion problem in [1] Wibling et al. propose the idea of abstract the flooding process of the routing algorithms verification. The main idea of this abstraction is that, if you are sure e.g. the flooding works, it can be used as a subroutine and do not need to be verified. Of course there are risks involved in such approach, and it should not be used for any process inside the network. When using this subroutine call, subtle errors may pass unnoticed through the verification. However, the authors argue these risks are minimized by the analysis of the process needed to formulate the model. Again, in model checking, there are always a tradeoff between the abstraction and the need into handle the state explosion problem. One must pay much attention when abstracting the protocol functions to do not make an incomplete verification. On the other hand, the state explosion is a real problem and can prevent the entire verification.

Wibling et al. have a very interesting and consistent work. However, as a drawback, they just verify LUNAR [70], their own protocol. Even though most of the techniques proposed by them are general, the lack of proof of a third part algorithm would provide better indices that the technique is really general.

Other point to observe is that some of the general assumptions they made about the network, only bidirectional links are allowed, may be too strong for some protocols. Other general assumptions made, the messages are delivered in sequential order and each node in the network can only receive and handle one message at a time. For some protocols, that do data aggregation for example, these assumptions are so strong that can even prevent the protocol verification. Regarding the state of the nodes, they don't have persistent memory, so if a node goes down, it loses all its states. For some protocols this may not be true, for example, some protocols use old data to perform optimizations. The most interesting fact is that, from our point of view, none of such limitations are really fundamental for the process as a whole and could be easily removed.

3.9 Counter-Example Based Predicate Discovery in Predicate Abstraction. Formal Methods in Computer-Aided Design

Das and Dill [43] propose a way to discover quantified predicates automatically from the model. They use this technique to prove the absence of loops in a simplified version of AODV. The initial predicate set is formulated in a manual step where conditions on next node pointers, hop counters, and existence of routes are constructed. The method successfully discovers all required

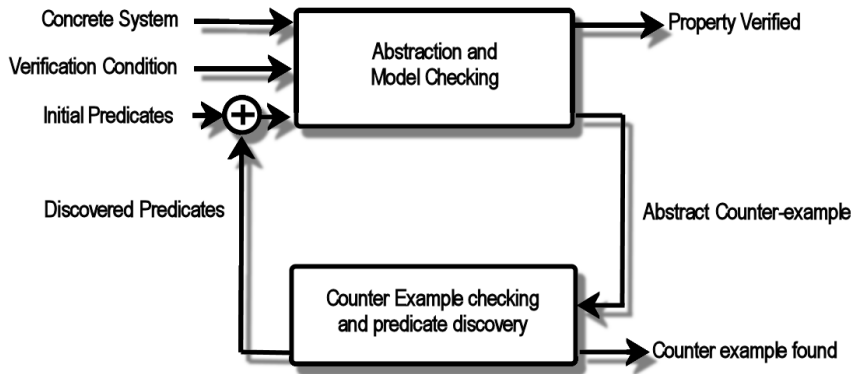


Figure 3.8: Predicate Abstraction

predicates for the version of AODV considered [1].

Unfortunately for the general case, the problem of finding predicates to an unbounded system is intractable. However, the authors claim that presented technique, Predicate Abstraction, is an efficient way of reducing infinite state systems into more tractable finite state systems. A finite set of abstraction predicates, defined on the concrete system, is used to define the finite-state model of the system. The states of the abstract system consist of boolean assignments to the set of abstraction predicates, that is, to each predicate is assigned a value true or false. The abstraction is conservative, meaning that for any property proved on the abstract system, a concrete counterpart holds on the actual system [43].

The problem of proving arbitrary safety properties of a transition system is undecidable, so the technique may fail for some cases. However, for other cases, given a pre-selected set of predicates and making some other assumptions, it is possible to prove that the system satisfies a safety property. On the other hand a failed proof may either indicate that the property is violated, or that the abstraction is not precise enough to complete the proof. An analysis of the results is often required on the failure case.

The work also introduces a way to find useful predicates automatically by diagnosing failed proofs. This is a huge step once, in the most of the previous works on predicate abstraction, the predicates were either assumed to be given by the user or extracted syntactically from the system description. The task of finding the right set of predicates is hard and often a trial-and-error one. Rarely the predicates present in the system description are sufficient so new ones must be added. The challenge in search for useful predicates resides in avoiding irrelevant predicates, since the cost of checking the abstract system normally increases exponentially with the number of predicates [43].

The schematic model of the authors proposal is shown in Figure 3.8. The upper block is a tool described in [74]. The system by itself was implemented using Binary Decision Diagrams (BDD). The tool receives the set of abstraction predicates, the verification condition and the concrete system description. With this an approximate abstract model is created. This abstract model is model checked and the abstract system refined appropriately if too inexact. This process finishes either when the verification condition verified or when an abstract counter example trace is presented. The lower block represents the search for the new predicate.

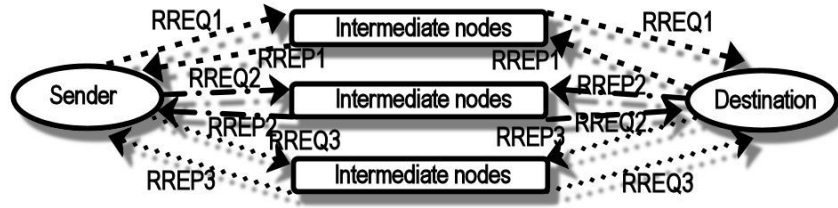


Figure 3.9: Chiyangwa Linear Topology Model

On behalf the work of Das and Dill Shuvendu et al. [75] state that predicate abstraction, with locally quantified predicates, require complex quantified predicates to construct the inductive assertions and these predicates are often as complex as invariants themselves. This compromises the real use of such technique. Other critic to the model is that the authors state that they cannot find a way to treat timeouts. This greatly restricts the range of protocols that can take advantage of the technique.

3.10 A Timing Analysis of AODV

Chiyangwa and Kwiatkowska [3] focus their work on the timing aspects of AODV using UP-PAAL. They build a timed automata model and considered the effects of the standard protocol parameters on the timing behavior of AODV. The authors investigate properties such as timely route discovery and messages delivered in a specific time period. The used timing automata can be found at [71].

With their model Chiyangwa and Kwiatkowska found that AODV routes rely in static parameters that can lead to failures when a network grows dynamically in size. The failures may be either in discovering a route that actually exists or in delivering data packages to a destination. Deriving from the results of this study the authors proposed the use of adaptive route timeouts to avoid these problems.

To work on the time aspects of the protocol the topology evaluated is nearly the same. It is always linear topology where the source is the node 0 and the destination is the node $n - 1$. All other nodes involved are sequentially placed between the source and the destination. The main reason to evaluate this singular topology is that the authors were focus in timing aspects and in finding the maximum network diameter. The chosen topology makes such evaluations easier and cleaner. Thus, the method model just three different kinds of nodes, source, destination and intermediate node. Each one of these node are modeled to accomplish a specific function.

To allow a simple instantiation of different number of intermediate nodes the authors propose the formulation of an n_nodes node, which combines n nodes linearly into one multiple node. Figure 3.9 shows the proposed Chiyangwa and Kwiatkowska linear topology model with the three routing discovery attempts proposed in AODV running in parallel.

The maximum network size for AODV was reached with twelve intermediate nodes. Given the expiration of the route lifetime of AODV, packets cannot be sent in longer routes. The authors also proved that in a network with seven intermediate nodes an established route may expire before the data packet transmission. These values in the real world can be even lower, once

to be fair the work did not consider message delays and message loss. Such kind of knowledge is of fundamental importance for the protocol creators, once it shows the limitations of their algorithm.

On [71] Chiyangwa and Kwiatkowska state that, because the state space explosion problem, they find fifteen intermediate nodes a limit for the feasible application of their approach on AODV. Fortunately, on this case, only twelve intermediate nodes were enough to find the problem. However, this is a point to observe, if applying this technique, in other contexts. Being unable to detect a problem can mean either that the problem really does not exist or that it is beyond the technique feasible implementation limit.

Other point to be aware about this work, is that to find the network diameter with the linear topology is an iterative process. One must run and re-run the model, increasing the size of the network, to find the protocol limits.

3.11 Topology Dissemination Based on Reverse-Path Forwarding (TBRPF): Correctness and Simulation Evaluation

Richard Ogier in [44] manually prove the correctness of the Topology Dissemination Based on Reverse-Path Forwarding TBRPF routing protocol [72]. Since TBRPF consists of two modules, the routing module and the neighbor discovery module, the author correctness proof is presented for both modules separately.

To prove the correctness of the routing module Ogier states the *theorem 1* "If no topology changes occur after sometime t_0 , then within some finite time after t_0 , the source tree T computed by each node will contain minimum hop paths to all reachable nodes". Then through induction, over the number of hops, Ogier proves that the routing module works optimally in terms of the least number of hops.

The discovery module works through differential HELLO messages, which report only changes in the status of links. Such hello messages are not reliable. Thus to prove the correctness of the discovery module Ogier needed to verify three possible cases: when the communication is bidirectional, unidirectional or lost in both directions. Ogier manages to prove that the neighbor information exchanged is sufficient for the functionality of the protocol. However the proof is neither easy nor valuable for other protocols. The results and procedures used stand for TBRPF and only for it. Other point that must be taken into account is that when verifying a protocol all cases must be considered and this, when performing a manual proof, is even harder.

3.12 Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad Hoc Networks

Using CPNs Kristensen and Jensen [24] verify the Edge Router Discovery Protocol (ERDP) an extension of the Neighbor Discovery Protocol (NDP) [73]. The process is iterative and both, a conventional natural language specification and a CPN model are developed. This work is

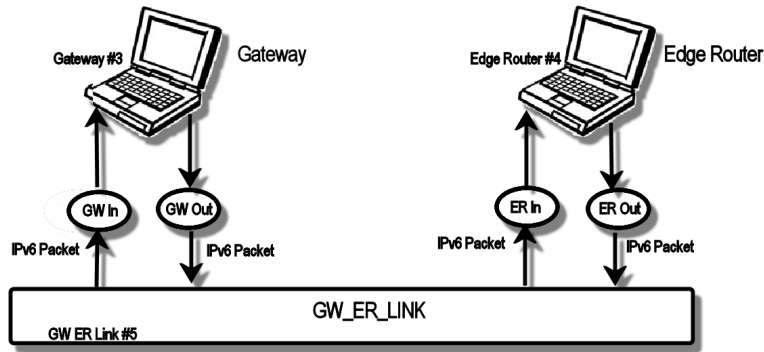


Figure 3.10: Kristensen and Jensen model of network architecture verified

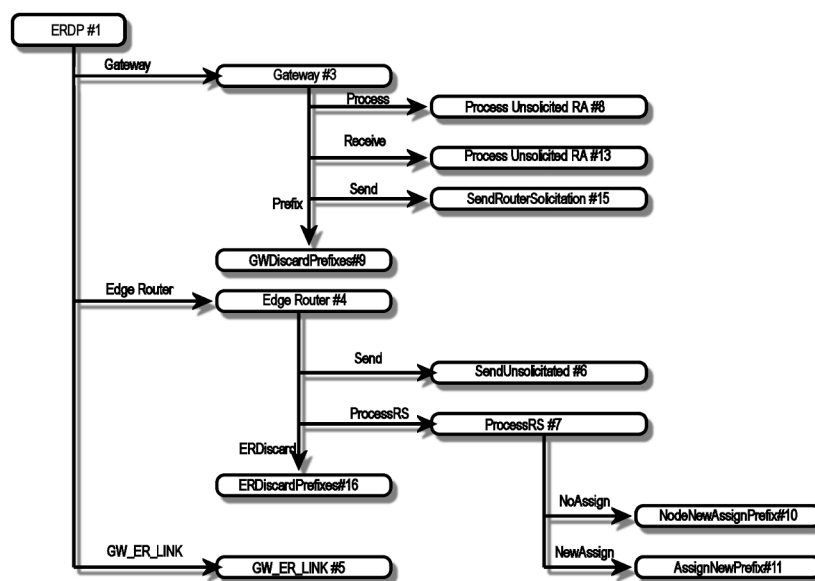


Figure 3.11: High view of the hierarchy page of the CPN model of Kristensen and Jensen work

mainly concerned with the state space explosion problem and the logical correctness of the algorithm. They do not intend to verify performance or any other non-functional aspect of the protocol. Thus on their verification Kristensen and Jensen uses just two nodes, a gateway and an edge router connected through a link. With this simple topology they mainly focus on the verification of the basic operation of the ERDP. Figure 3.10 presents the architecture, composed by the Gateway, Edge router and Gateway-Edge router link. Figure 3.10 makes references to the pages of the Figure 3.11 that describes the high view of the CPN model.

One interesting aspect of this work is that it is a partnership between academia and industry and it was a part of the development of ERDP. Researchers from the academia developed the model and people from industry implemented it. On this process many failures were found because the synergy of the two groups. With two revisions revealed 24 issues on the protocol and its specification. An interesting data from this study is the approximate efforts spend on it. The total amount effort of constructing the CPN model and conducting the state space analysis, one of the concerns of this work, was approximately 100 man-hours. This data is relevant and

not very common in such kind of work.

The concerns about the state space explosion of Kristensen and Jensen is completely fair and justifiable. However the use of only two nodes penalizes the approach and even the results it may reach. For example, any problem with the concurrent aspect of the protocol is probably completely lost with this simplistic approach.

Chapter 4

Methodology

This work is built over the one presented in [3], and it is a step-by-step procedure to verify routing protocols for wireless ad hoc networks. The focus of the verification method is model checking. Typical formal verification approaches applied to routing protocols for MANETs, such as Wibling et al. [1], and Chiyangwa and Kwiatkowska [2], use either a specific network configuration or a given number of nodes on the verification. The problem with these approaches is that mobile ad hoc and sensor networks are dynamic systems. Therefore, the correctness proof of a particular configuration does not guarantee the correctness of the protocol with respect to other configurations. This work is grounded on a complete different principle. It does not model any particular network configuration. Instead of that, it proposes that one should model all the possible implications caused by network configurations to the behavior of the routing protocol for MANETs. In other words, the verification should model all the possible relations among the nodes. This is a key observation in our technique.

When applying model checking verification a designer wants to determine, in an automatic way, if a given model M presents a defined property P . Both, M and P are provided by the protocol designer and precisely defined. M is composed by the finite set of variables V , $V = v_1, \dots, v_n$, the set of initializations I , where it is applied $I(V)$ or I is a condition over V , and a set of transitions T , $T(V, V')$, where V' is the new value for the variable V after the application of the model step. The model checking tool uses then M , to build the set of all possible system states, we use SPIN model checker. Let $G = (V, I, T)$ be the set of all states, and $P = (V)$ the property to verify, the tool must than search if P can be satisfied starting with I and applying T a finite number of times. If M model all the possible relations, than G contains all the possible outcome system states.

The method is independent, and could be implemented with any formal verification tool where ground principles, described below, could be modeled. We use SPIN model checker since it is freely available and matches perfectly our needs. All the snippets of code for the next sections, and on the appendix, are described in PROMELA, the SPIN programming language. When in the text we refer to random initialization that means that the set of initial values is defined, but it is up to the tool decide which values to test, normally all. For example, when we make a random initialization of a boolean value, we set the possibility for the tool to choose both values, if we are doing a full verification the tool will need to test both paths. However, if we are using a random path verification just one of the values will be chosen by the tool, in a

random way. Again, we use the term random initialization to define all the possible values.

4.1 Limitations

In contrast to the work presented in [2], this method focuses mainly in qualitative aspects rather than quantitative ones. We are interested in identify routing loops, packet delivery failures, unexpected reception of messages and pathological cases not treated in the original protocol specifications. However, quantitative aspects, such as max/min number of messages exchanged among nodes, behavior in a particular topology or timing aspects, are not the focus of the method and are not covered by it. These problems are handled by other authors.

4.2 Ground Principles

This section presents some of the principles that work as basis for the technique. To decrease the complexity of the models and avoid the combinatorial explosion problem this work proposes that the verification must follow some ground principles. They are: topology abstraction, node position and lower layer services.

4.2.1 Topology abstraction

Many of the techniques described at Chapter 3 consider the network topology as an integrant part of the model and the verification. Any proof that takes into account a specific topology just proved that the protocol is correct or not regarding particular evaluated topology. We advocate here that, for the set of properties we want to verify, the topology is not a relevant factor, since the errors are based on the relation among the nodes, not on their specific positions. So, instead of enumerating all the infinite possible topologies, it is better to avoid it and focus on the node relations. In order to do that, the method proposes the use of three kinds of nodes, namely, source (S), destination (D) and intermediate (N) nodes. Let R , $R = (Source, Destination)$, be the set of all possible relations among the different kinds of nodes, $R = \{(S, D), (S, N), (S, Null), (N, S), (N, N), (N, D), (N, Null), (D, S), (D, Null), (D, N)\}$, where $Null$ represents the case where the message is lost or corrupted. If the model M contains all the relations defined in R , and R contains all the possible and relevant message forwarding cases, then all the possibilities will be presented in G , and thus in consequence in the verification.

Representing the set of all possible intermediate node relations, $(S, N), (S, Null), (N, S), (N, N), (N, D), (N, Null), (D, N)$, enables the verification of all the situations where transmission related properties could arise. These relations represent, specially (N, N) and $(N, Null)$ represents all the possible transmissions among intermediate nodes. These relations also represent, implicitly, the set of all intermediate nodes and all possible effects of interconnections among them. For example, a node with a neighbor and a packet transmitted successfully, a node with a neighbor and a lost packet, and so on. The key aspect to make sure the topology abstraction works is to model all the possible relations and this is especially true for the intermediate node.

With this process the topology becomes irrelevant since the relations among intermediate nodes will model the possible relations, not the network topologies. Consider $C \subset V$, where $C = \{c : c \in M \text{ and } c \text{ controls the message flow}\}$. C is the set of all procedures used by the target protocol to control the flow of information through the network, for example, c could be the last received message counter. Assume that $\forall c \in C, c = \{0, 1\}$, if $I(C)$ is random, the tool will generate all possible initial values, each of the two possible values for each c will be represented in G . If G contains all the states, the search for P over G is granted even without considering the topology. This different way to face the network is what ensures the topology independence to the method, once it does not rely in any particular topology. This greatly decreases the complexity and, in consequence, the effort required to formally verify the dynamics of routing protocols. One of the biggest problems for other methods, that use the network topology in the verification, is that they are somewhat bounded by the size of the network. The state space explosion problem makes unfeasible the verification of protocols behavior for large networks.

4.2.2 Node position

Position awareness is one of the criteria for classifying routing algorithms. However, with the topology abstraction, the nodes position are irrelevant since what we represent are the possible kind of nodes relations. This stands even for position aware algorithms, since we are concerned with the possible relations, not specific configurations.

This principle, even being counter intuitive at first sight, holds exactly in consequence of the previous principle, the topology independence. When we focus on the possible relations, defined in R , they are the same whether one considers the nodes position or not. For example, if two nodes are connected, normally, their physical positions do not matter. However, if the verified protocol is a geographic one and uses, in some way, the nodes positions, the nodes relations must also be modeled. For example, LAR1 protocol uses the intermediate nodes geographic position to determine either if they should rebroadcast the message or not. In this case instead of trying to model and verify all the infinite positions in the three-dimensional space one can just add to M the two possible relations, either the next hop is inside the re-broadcast area or not. Doing this, for this case, the need for geographic positions disappears.

4.2.3 Lower layers services

Layers others than the one where the algorithm in test is located should be abstracted, once they are not the object of study. The services provided by the lower layers should be modeled as available and trustable, unless some cross layer aspect is crucial for the protocol validation. In this case, the verification, for this cross layer aspect, should be done apart. This step is the application of the assume-guarantee paradigm [78, 79], since assume that the lower layers behave as expected and, if it is required we need to guarantee this assumption through verification.

The main idea behind this is to simplify as much as possible the protocol verification to avoid the state space explosion problem. Unless the verified protocol considers cross layer aspects, it is possible, without lack of generality, to consider just the interaction, or API, between the layers and ignore the way the other layers work. For the routing layer, for example, the MAC layer should work as a procedure call. E.g. the protocol requires a message to be sent to a specific

node, either this service works, the message is delivered, or not, for the routing layer it is all that matters. However, if the protocol consider, or uses, some specific characteristic of the MAC layer, then this behavior, at the MAC layer, must also be verified to ensure its correctness.

4.3 Modeling

In accordance to the technique described here, the algorithm one wants to verify should be modeled following some guidelines, in respect to some specific aspects. Following these guidelines enable the designer to verify aspects that are not possible using other techniques. This section presents these guidelines and also provides a clear explanation why such guidelines are important to the whole process.

4.3.1 Communicating channel

One of the key aspects to enable the proper verification is the modeling of the communicating channel. The channel should be available in a random way among the three defined kind of nodes or no node at all. Let K be the set of packets defined in the target protocol, $K \subset V$ and $K = \{\forall k \in K : k \rightarrow O \vee k \rightarrow N \vee k \rightarrow D \vee k \rightarrow Null\}$. Where the relation \rightarrow stands for "from the present node may be delivered to". This means that any node, or even no one at all, may receive the packet. This guarantees that all the relations will be verified in the end.

However, the way the protocol is designed must determine if the packet will actually reach the destination or not, regardless the way the channel is modeled. This point is important since it, in the deepest analysis, what enables the intermediate node to represent all the possible relation among intermediate nodes.

In other words, is the channel model the main responsible for the topology independence. The possibility that any node, and even no node at all, may receive the communication implies that all the possible scenarios are present in the model. It is important to call attention to the fact that there is no topology involved, just the plain and simple relationship among nodes.

4.3.2 Flooding representation

Relying on this communication channel model two messages are enough to represent all existing relations in a flooding. What happens here is that if $I(K)$ is correctly done, two messages on the network will represent a Cartesian product of K over K , $K \times K$, where all the relations among the same packets over the kind of nodes will be represented. The need for two messages to represent a flooding came from the fact that at least two messages are needed to represent concurrent transmissions. I.e. the case where two nodes send the same message to a given node at the same time.

However, when verifying a protocol, the designer must make all possible relations reachable. If the relations are not reachable, by a modeling mistake, the verification results may be compromised.

4.3.3 Mobility

From the nodes point of view, the main consequence of the mobility is the occurrence of broken links. So, as the model represents all possible relations among nodes, including connected and disconnected nodes (i.e. broken links), the mobility is also modeled.

The changing in the neighbor nodes may affect the way some protocols work, i.e. the need of attach to a new cluster head, or send position updates. The protocol designer must be aware that even not considering the nodes position the behavior of the protocol, regarding mobility, can and indeed should be modeled through the broken links.

4.3.4 The network

More important than identify all possible topologies is to identify all possible effects of different topologies and node states to the routing protocol. For instance, what can happen to a message after it is received by a node? Some possible scenarios are: it can be lost because of a node failure, it can be transmitted to a neighbor node with a checksum error, it can be transmitted successfully, and so on.

In this way, reviewing and modeling all the possible network behavior, one can increase the chances of having a successful protocol verification. These relations are also extremely important, even more when analyzing the verification results to see if the failure scenario found is really possible or not.

4.3.5 Internal and external behavior

The division of the protocol into internal and external behavior is an instance of compositional reasoning [80]. Composition is a *divide and conquer* approach [81] where the target system is divided into small components that are verified separately. However, it is important to notice that the composition in this case refers to the division of the protocol behavior and not functionalities. The protocol behavior should be divided into internal and external, and each part verified separately. Internal behavior refers to how the protocol handles data and controls messages internally to the node. In other words, the actions the node implements when it receives, or sends, a specific message.

The external behavior refers to how the whole network reacts to the messages. Both models should be independent and modeled in such way that the internal behavior could act as a procedure of the external behavior. The idea is model the two different behaviors as if they were one part of the other. However, when we break the behavior not only we manage to focus in the verification of two different aspects separately, but also we decrease the model complexity. Both behaviors start, in general, with a packet and its relations. The initialization of the packet data and its relations should be random to guarantee the coverage of all relations.

The internal behavior is the one where, normally, the largest part of the protocol will be represented and thus where the biggest part of the errors will be identified. This is expected once the internal behavior should be exactly how the protocol behaves inside the node and thus it is the way a real implementation of the protocol would work. However normally not all the network expected behavior can be inferred from the internal behavior implementation. The

external behavior main purpose is then to verify if the network reacts as expected and if the desired network properties hold.

When creating a protocol designers normally try to create a general network behavior based on the nodes actions. For example, the OLSR protocol routing information dissemination procedure explained in detail in chapter 5. The node behavior of choosing a set of multi-point relay (MPRs) and send the messages through them can be verified through the internal node part. However, for example, the expected process of message spreading, that should resemble a tree creation, should be verified modeling the planned external behavior. To make the external behavior as close to the real one as possible it should be implemented using calls to the internal behavior. These calls should just return, in a random way, the possible results, but not implement the verification of the called parts. In this way one decreases the complexity of the model at the same time keeps the models consistent and the verification coherent.

4.3.6 Information modeling

Every information regarding the verified protocol should be modeled as a variable and, as far as possible, randomly initialized (e.g., package type, packet time-to-live (TTL), and table exchange trigger). When we say random initialized, we mean that the choice of each of the the total set of possible values is random, but the set of values must to be defined. For example, when possible, such information should also be modeled with boolean variables. Both values should be defined, but the choice of which of the values will be taken during the verification, is on behalf of the tool.

This way of modeling the information, decreasing the set of possible values, and, for example, modeling the TTL as Boolean, is a form of abstraction [30], since we will not be representing the whole set of possible values for a $v \in V$. For example, if the packet has a TTL, not all its values need to be verified: if the TTL was reached, or not, is normally enough. In PROMELA this can be represented as:

```

bool ttl;
if
  :: (1) -> ttl = 1
  :: (1) -> ttl = 0
fi;

```

For this specific example the variable `ttl` may receive the value 1 or 0, meaning the time to live of the message expired or not. So as the attribution is done in a random way, both cases are possible, the verification tool will be forced to verify both cases, what to do when the `ttl` expired, and when do not.

4.3.7 Procedures Abstraction

To avoid the combinatorial state space explosion a protocol should be simplified as much as possible, while this does not compromise the verification results, of course. Over simplifications may often lead to wrong conclusions, so the amount simplifications over the protocol should be carefully applied. This is, again, the application of the abstraction paradigm [30]. However, here

we are interested in the simplification of the algorithms procedures, not variable values. $\forall t \in T$, the designer should verify if t cannot be reduced to a simpler representation in V , preferably as a boolean.

As an example of possible simplification, suppose a protocol that uses the Dijkstra or Floyd protocol to find the minimum path. For the verification purpose, it may be enough to model the shortest path as a boolean variable, either the protocol finds the path or not. This may be enough because when verifying a new protocol, the designer is normally more concerned about his own algorithm than the shortest path one. However if it is not the case, if the protocol changes something, or uses a sub phase of the minimum path algorithm, this behavior also must be present in the model, and just one boolean variable may not be enough.

4.3.8 Model

The development of the protocol model should start simple and one should increase the model complexity appropriately. With this approach, basic problems can be identified earlier and possible solutions can be quickly proposed and validated. This also allows the protocol designer to stop verifying the protocol whenever it reaches a defined goal or a reasonable model complexity.

The verification, as it is any test process, can extend indefinitely since at any time one may think of new things to verify and test. It is much more realistic to think that the verification phase of the protocol, in the real world, will be bounded for time/resource limitations. In this way if one starts with a complex and extremely detailed model, all the work can be wasted if the limitations are met. It is wiser to have useful results with simpler models and be able to profit from the verification results of these simpler models.

4.3.9 Analysis

Every time a property is verified and the tool presents a response scenario, the designer must analyze whether the result is a fault on the protocol or on the model. For example, simplifications can introduce errors in the model in some way that some of the found properties even being present in the model are not possible in the real world. $\forall p \in P : (p \leftrightarrow G) \Rightarrow a$, being P the set of properties to be analyzed, the relation \leftrightarrow , that produces the answer a , must to be analyzed.

This is a crucial step and, unfortunately, as it is the actual technological state of the art, cannot be done automatically. It is required a human operator to reasoning over the scenario and decide whether the error may or may not occur in some way.

4.4 Algorithm

This section presents a description of the verification methodology in an algorithmic form. Using this algorithm one can easily apply the methodology as it highlights the most important points of the method and presents the tasks in an organized and coherent sequence.

The tasks are intentionally presented in high level to provide a useful abstraction and freedom to the designer to choose the best approaches and tools to fulfill the required steps.

The first step, acquire the needed information is maybe obvious but it is surely enough, in conjoint with the next two, some of the most important steps of the method. All the other steps,

actually the entire verification process itself is useless if one does not understand the protocol. If the protocol is not clear for the person who is performing the verification it is possible that important procedures and characteristics of the protocol may never be verified, or even worse a completely different protocol may be verified instead.

The second and third steps should occur as many times as needed to one to have a consistent representation and reach a complete understanding of the protocol. Creating a representation of the protocol in an algorithmic or pseudo code provides to the designer not only a better view of the protocol but force them to think in the exception condition. Normally the designers, when creating a protocol, tend to pay attention to the big issues and are somewhat more careless with the exception cases. In this way these cases, which are not fully defined may hide situations where the protocol may fail.

Algorithm 1 Verification Methodology Steps

```

1: Acquire needed information to model the protocol;
2: repeat
3:   Create a detailed pseudo-code or finite state machine of the protocol;
4:   Compare carefully all cases described in the protocol with the pseudo code;
5: until Pseudo code is consistent with the protocol
6: for Each kind of packet do
7:   for Each kind of node do
8:     Specify the semantics of the packet to the node;
9:   end for
10: end for
11: Divide the protocol into internal and external behaviors
12:   Internal behavior: describes the message flows and behaviors for the node;
13:   External behavior: describes the behaviors related to the node interactions;
14: for Each behavior do
15:   Create an algorithm or an state machine representation to ensure the validity of the abstraction;
16: end for
17: Model the External vs. Internal interactions
    // The internal behavior should be modeled as if it was a routine call.
    // In this way the external behavior becomes independent of the internal behavior.
    // Ideally, the external and internal behaviors should be independent;
18: Build a simple model based on the internal and external behavior machines
19: while The desired model complexity was not reached do
20:   Analyses and verify the model
21:   for Each error found on the model do
22:     Verify whether the error is due to a protocol failure or a modeling failure;
23:     Find a solution for the problem;
24:     Model the solution;
25:     Test the solution;
26:   end for
27:   Increase the model complexity;
28: end while
29: Identify and isolate verified procedures to be used in other protocols.

```

Steps from 6 to 10 also intend to provide a better understanding of the protocol through the organization of the protocol messages. These steps make it easy to generate the model once they summarize the messages and their meaning for each node. It is crucial to enumerate accurately all the cases for all the nodes and messages. In this case a table having the three kinds of nodes as columns and the packets as lines, or vice versa, is a good exercise since forces the designer to fill all the gaps for all the combinations.

Lines from 11 to 13 advise the methodology user to divide the protocol into Internal and

External behaviors. This means that, normally, the designer is advised to verify its protocol using two different models. For some cases maybe the external view can be overlooked, if one can represent efficiently all the protocol interesting behavior with the internal view. However, at this step the designer is not creating the model yet, but just thinking which routines could be better represented in each model. The steps 14 to 16 are the responsible for verifying if the division is feasible and coherent. Creating an algorithm or a state machine forces the designer to really understand and validate the division he/she made.

When creating a model for verification one should start with the simplest model possible, this is advised in the steps 17 and 18. Using this tactics one can profit from earlier results at the same time understands better the protocol he/she is verifying. The real verification process occurs in the loop that starts on step 18 and ends on step 28. The verification is defined here as an iterative process. One should get the simplest model, verify the desired properties over it, step 20. If it is not detailed enough the model should be refined, and its complexity increased, step 27. If any error was found on this model, step 21, the designer should verify if the error is in the model or in the real target protocol, step 22. Either one of the alternatives, a solution for the problem should be found, step 23. This solution should then be modeled, step 24, and finally this solution should be verified to guarantee it is feasible and that the protocol is now free of the previously detected problem.

The step 29 intends to create a library of verified processes to enable easy design and verification of newer protocols. If one could use a library of verified procedures there is a possibility the quality of the new algorithms and protocols would increase significantly.

Chapter 5

Case Study

This chapter presents the methodology applied to OLSR protocol [33], using SPIN model checker. Notice, however, that any model checker that allows the modeling of non-deterministic channels can be used. The examples presented here are in PROMELA, the SPIN language. Another important observation is that, even though, OLSR has newer and more precise descriptions [34], in order to meet our objectives, given its simplicity, we use the original version [33].

The structure of this chapter follows the algorithm presented in Section 1. This toy example exemplifies how to apply the technique in a real protocol finding real problems on this protocol.

5.1 Modeling

5.1.1 Understanding the protocol

The Optimized Link State Routing Protocol (OLSR) [33] is proactive protocol. In order to disseminate routing information, a node sends both hello messages to its neighbors and topology control (TC) messages to a set of selected nodes that, in turn, re-broadcast them to other MPR (multi-point relay) nodes. The broadcast information includes the node address and a list of distance information of nodes neighbors. With this information each node builds its routing table, using a shortest path algorithm. In this way, the node can create a route to every other node in the network. If a node receives a duplicated packet it discards it rather than retransmitting it. However, the key concept of OLSR is the multi-point relay (MPR) nodes. OLSR relies on MPR nodes to retransmit information in an organized and smart way. The MPR nodes are chosen among the 1-hop neighbors in such a way that they are the minimum set that covers all the 2-hop neighbors.

More specifically OLSR is composed of five steps: Symmetric neighbors detection, Multipoint relays, Optimized flooding, Partial link state and Optimal route calculation. The Symmetric neighbor detection consists of nodes periodically broadcasting hello packets, what advertise the heard nodes set. Based on the hellos nodes perform the Multipoint relay selection, i.e. every node selects its multipoint relay set (MPR). After that the control are broadcast through Optimized flooding, only the multipoint relays forward broadcasts. Through the MPRs all network nodes receive, but only a subset retransmits. The periodic topology control (TC) messages are broadcasted through the MPR the result is a much smaller set of nodes rebroadcasting the TC

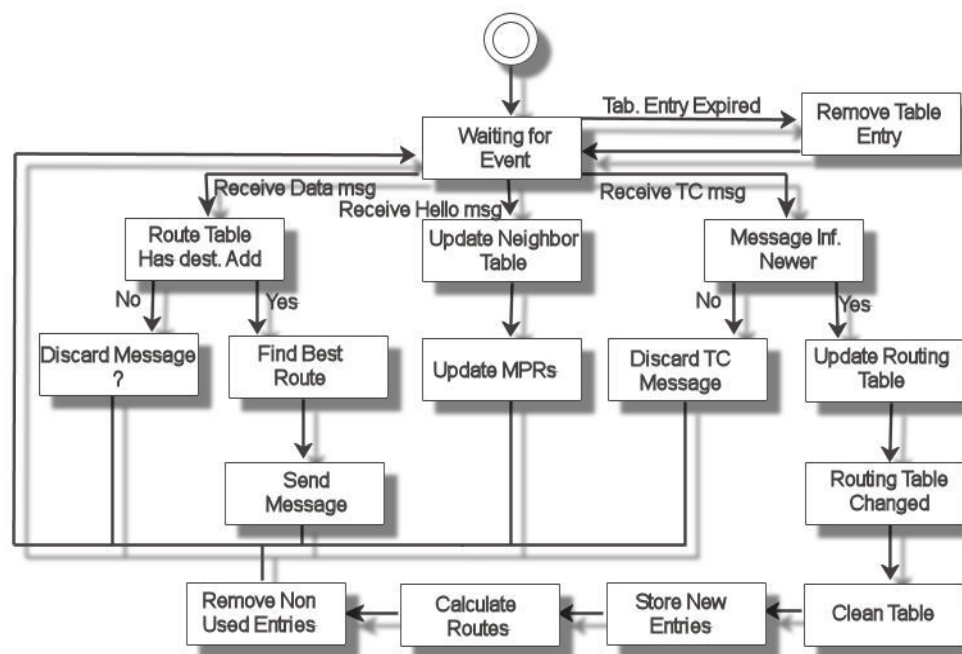


Figure 5.1: Simplified OLSR intermediate node state machine diagram

messages. The Route calculation is based on the MPRs. The Nodes know their neighbors and all the network MPR links. The route calculation occurs on partial topology knowledge. From time to time the tables are erased and recalculated with basis on the last known information.

5.1.2 OLSR State machine

To better understand how the protocol works it is often useful to rewrite it in an algorithmic form or translate it to a state machine. Just doing this sometimes one can find errors or flaws in the protocol description. Some of the most important errors found in AODV by Obradovic et al. [2] were exactly specification errors. These specification problems occur, in some extent, often because the protocol designers are normally much more concerned about the main protocol ideas and are somehow more careless describing the details and smaller, but important, design decisions. Figure 5.1 shows a simplified diagram for the OLSR intermediate node, suitable for an early verification model. In the diagram the discard message box, when there is no route for data packets, has the symbol "?" because the protocol description has no explicit reference about this specific case, so we will need to infer what to do.

5.1.3 Messages and kinds of nodes

OLSR defines three different kinds of messages: hello, topology control (TC) and data message. Each one has its own purpose and semantics for each different node. Table 5.1 presents these messages and their meaning for each kind of node.

Node	Hello	TC	Data
Origin	Spread information of links among neighbors	Spread information about the network table	Share upper-layer protocol information
Intermediate	Update its own neighbor table, find MPRs	Update routing tables and rebroadcast	Find new route and rebroadcast
Destination	This role does not exist, act as intermediate	This role does not exist, act as intermediate	Send msg to upper layers

Table 5.1: OLSR Messages and their semantics for each node

5.1.4 Dividing into internal and external behaviors

The main part of the verification is done by the internal model behavior. This is expected since if we model the behavior of each node in respect to each different message, it is exactly how the distributed protocol should behave in the network. However, sometimes some behaviors can only be captured globally and, in this case, the external view is quite useful. For the OLSR this is clear when we observe the creation of the tree of MPRs and the transmission of messages through it.

For the OLSR protocol the modeling of the external behavior was worth it because it showed that the source node may receive the same TC packet it has generated. This may occur because the source node may be the MPR, or may be in the vicinity of an MPR of the node that is retransmitting its TC message. We cannot characterize this as an error, but at least for us this is somehow surprising, once the whole structure is intended to decrease the number of messages sent and it is always represented as a tree without loops.

5.1.5 Modeling the channel

One of the key aspects of the methodology is doubtlessly the channel modeling. Modeling the channel in a non-deterministic way makes the results more general and, indeed it is what makes it possible for the intermediate node to act as a cloud of nodes. This simple observation greatly decreases the complexity of the models and allows their verification independently of any specific topology. The channel could be modeled in PROMELA as follows:

```
mtype = {RouteRequest, RouteReply, DataPacket};
        /* type, origID, destID, TTL, inf timestamp */
chan medium =[nnodes] of {mtype, byte, byte, bool, bool};
```

However, the most important aspect of the channel is the way nodes receive messages from it. The choice must be random and any node should be able to receive the message at any time. The protocol behavior will be in charge of handling the messages according to its specification.

Another important channels characteristic is that it should be able to hold more than one message. OLSR is heavily based on a controlled flooding, but flooding nevertheless. If all relations are modeled, putting at least two messages in the channel, it enables the verification

of all possible node states, even concurrency.

5.1.6 Creating the model

To model a new protocol it is better starting with the simplest model as possible, Algorithm 2 shows a first version of the OLSR model. After that the model complexity can be increased slowly until the desired level. Building the model in this way has several advantages. First, it helps a designer to better understand the protocol. Second, if one finds an error in the protocol, it is easier to isolate the problem and find a solution for it. Third, building the model in this way makes it easier to debug and finding out errors on it. However, the most important advantage is to have results sooner. Starting the process by building a complex model is probably a much more difficult task and, sometimes, people simply give up in the middle of the work. It is better to have results sooner and being able to stop at some comfortable point, than to have a complex model, which most surely will even drive the model to the state explosion problem and, worse, without any result.

The model should be as simple as possible, while this does not compromise the protocol verification. For example, the condition of the OLSR timer to send a hello message can be modeled as a boolean variable, i.e. either it is time to send the hello message or not. In this case there is no need for modeling a real timer. This trigger could be programmed in PROMELA as:

```

bool sendHello;
if
  :: skip -> sendHello = true
  :: skip -> sendHello = false
fi;

```

Every variable, as much as possible, should be randomly initialized within a defined interval. This increases the number of verified cases and makes the verification more independent and broader. Once a variable is initialized randomly, all cases related to that variable will be verified automatically. Of course, in the hello message example above, we are assuming that the message sending procedure represents an independent event. It is neither triggered nor affected by other events. The drawback of this approach is that the model may become so broad that even cases that can never occur in the real world may be present in the model, and, thus, leading to false positives that need to be analyzed and discarded. Although this is probably better than missing a real world case in the model. Algorithm 2 shows an example of a first possible version of a PROMELA code for the verification of OLSR algorithm.

5.2 Verifying the model

In SPIN the verification is done based on propositions represented in linear time temporal logic (LTL). To make the verification easier, each protocol scenario, especially the ones that can lead to errors, should be identified with a different variable, normally a boolean one. Building the model in this way enables the creation of simpler and straight forward LTL formulae. To verify

Algorithm 2 Example of a simple first version of the OLSR model

```

1: #define nnodes 3 /* Intermediates + Origin + Destination */
2: chan medium = [nnodes] of { mtype, byte, byte };
3: mtype = { Hello, TopologyControl, DataPacket };
4: mtype = { Origin, Destination, Intermediate };
5: bool packt = false;
6:
7: proctype nodes() {
8:     bool ttl;
9:     byte type;
10:    byte node;
11:
12:    if /*build the packet*/
13:        :: skip -> ttl = true /* reached the TTL*/
14:        :: skip -> ttl = false /* not reached the TTL*/
15:    fi;
16:
17:    if
18:        :: skip -> type = Hello
19:        :: skip -> type = TopologyControl
20:        :: skip -> type = DataPacket
21:    fi;
22:    medium!type(ttl);
23:
24:    if /*threat packet*/
25:        :: skip -> node = Origin /*acts as origin node*/
26:        :: skip -> node = Destination /*acts as origin node*/
27:        :: skip -> node = Intermediate /*acts as origin node*/
28:    fi;
29: }
30:
31: init {
32:     run nodes();
33: }

```

a proposition becomes just a matter of verifying the state of a variable. For example, the LTL formula to verify if the protocol fails to deliver a message could be:

$$\| (\text{!failDeliveryMessage} \ \&\& \ \text{PathExists})$$

5.2.1 Case study results

Using the technique and incrementing slowly the complexity of the model presented in Algorithm 2, one can find a series of errors in the OLSR protocol, some presented here. Table 5.2 presents the sizes of SPIN verification for different OLSR built models. It is important to highlight again that neither formal verification nor testing can guarantee that the system is perfect [11]. Indeed, we do not intend, by no means, to claim that the errors presented here are the only ones present in the protocol specification. However, what we do claim is that, for sure, at least these ones exist. As proposed in [8] the algorithm to recalculate the routing table, first cleans the entire routing table prior rebuilding it. It is not clear in the original work how this procedure is done; actually this can be seen as an incompleteness of the protocol description. However, if a data packet arrives at this time OLSR may raise an error because there will be no route available and the packet may be even discarded. This spin trace for this case is depicted by the trace at the Appendix, subsection "Trace - LTL:[]!dm, distinct case".

Implementation	Depth reached	# States stored	# Transitions	Memory usage (MB) for states
First version	7	34	46	0.003
Second version	10	55	67	0.003
Third version	18	637	637	0.033
Fourth version	37	61469	96637	3.442
Last version	89	2245174	3533252	196.987

Table 5.2: Sizes of the OLSR built models

Other problem that occurs in the OLSR specification is that, when a message arrives in a node, just after the link is marked as unidirectional instead of bidirectional, the control messages may be discarded. With this, possibly, not all two hop neighbors will receive such message. The problem here is that authors argue that the MPR nodes are enough to guarantee that all two-hop neighbors will receive the control messages, once they represent the minimum set to cover all two-hop away nodes. This statement may not hold if, for any reason, a node stops to act as MPR. In this case part of the network may be uncovered until another node takes its place.

Authors also argue that OLSR is resilient to a message loss. However, the protocol removes old entries from its tables if they are not refreshed in a defined amount of time. If the update message is lost, even if the route is still valid and able to deliver messages, the entry may be removed.

The last two OLSR problems presented here are the following. It has no explicit control for counter overflow stated on the main paper [33]. Thus, whenever a counter overflow occurs, the older information is kept on the routing tables instead of newer ones. This situation holds at least until the information entries are discarded by aging. This apparently is not an important problem, although, it can lead to another more serious problem that is a routing loop, at least for a short amount of time. The loop case may befall if a conjunction of over-flow and message lost situations occur leading to inconsistent routing tables among nodes.

Chapter 6

Application to LAR and DREAM Protocols

To validate the methodology, we use three different routing protocols for MANETs: LAR1, LAR2 [7] and DREAM [6]. The first two algorithms are geographic routing protocols and the last one is a link state based protocol. Such algorithms were chosen to show the usefulness of the methodology. The first two algorithms are well-established geographic routing algorithms with some well-known flows. The idea behind this is to know whether the methodology can be applied to geographic routing protocols, ignoring the node position, and whether the methodology is able to, at least, detect the known flows of such algorithms. Another point about these protocols is that they are flooding-based ones. We wanted to make sure our technique really works with this class of algorithms as well. On the other hand, OLSR is a well-known and cited algorithm, which became an RFC [16], and uses a completely different routing approach.

Without exception all evaluated protocols, LAR1, LAR2, DREAM and OLSR, presented problems. Some of them are well known, such as the inability of both LAR and DREAM to find an existing route. However, others such as the presence of a loop on both DREAM and LAR2, in the best of our knowledge were not previously known for these algorithms.

6.1 Evaluated Routing Protocols

The Location-Aided Routing (LAR) [7] routing protocol has two different variants, LAR1 and LAR2. Both are geographic routing algorithms but working in a quite different way. LAR1 defines a rectangular requesting zone defined by the two extreme points of the following diagonal: the sender position and the old destination position including a circular expected zone of this node. Inside the requesting zone the packets are flooded to reach the destination. In LAR2, instead of a rectangular expected zone, every node, when receives a message, calculates its own distance from the destination and verifies if such distance is greater than the one from the previous node, considering a δ threshold. If this distance is greater, the packet is discarded, if not the packet is forwarded.

The Distance Routing Effect Algorithm for Mobility (DREAM) [6] is based on two simple observations, distance effect and mobility rate. When a node needs to send a message to any

other node, it verifies the position, velocity and information time of the destination node on its own routing table. With these the source node estimates the area where the node is and calculates the angle α , which defines the destination node expected reachable zone. With this expected zone the source defines a triangular region between its position and the tangents to the expected zone. Then it sends a unicast message to the nodes within that region, which in their turn repeat the same procedure until the message reaches the destination.

6.2 Modeling

The properties verified in this work where loop occurrence, valuable packet dropped and packet not delivered. Each one of such properties was modeled as boolean variables and the occurrence or not of the property was the LTL formula verified. The scenarios were small and occupied from 1.4 to 1.6 MB of space for each property verified. All verifications used the default SPIN hash table state size, which is 218. The stored or reachable states for the protocols, with the verified properties, varied from 91 to 2489. The maximum longest depth-first search path for the verified properties varied from 17 to 90 steps. These data are indicative of how tractable the models became.

6.3 Results

Among the design errors found using the methodology we report: (i) fail to deliver messages in LAR1, LAR2 and DREAM, even though all three protocols use controlled flooding to deliver messages, (ii) loop scenarios in LAR1, LAR2 and DREAM, somehow unexpected for geographic routing algorithms; (iii) fail to deliver messages in OLSR, when a message arrives during a routing table recalculation; (iv) discard newer table information in OLSR and (v) control messages discarded in OLSR. Table 6.1 presents the sizes of the SPIN verification for the models of the verified algorithms.

Figure 6.1 shows a scenario where LAR1 fails to deliver a message, although there is a path between source and destination. This error will occur in LAR1 whenever the only path available to the destination passes outside the requesting zone, this was an already known failure that was confirmed through the use of the methodology. Figure 6.2 shows the scenario where the same deliver a fail happens for LAR2. For LAR2 the error occurs if any node in the path is farther from the destination than the previous one. This error was well known for LAR, however, this fail can occur also in DREAM, Figure 6.4, if the path is not inside the expected zone and this, in the best of our knowledge were not known for DREAM. In LAR1 the delivery failure may occur if there is a path outside the expected zone and no path inside it. This is interesting because even doing a flooding there is no guarantees that LAR or DREAM will effectively deliver a message. Another identified situation where LAR1 may fail delivering a message is when the expected zone and the origin node are aligned. When this occurs few, or no nodes at all, are found inside the expected zone, and, thus, the packet is lost.

In a broader point of view all three protocols may fail delivering messages because route concavity. In other words, if the only viable route passes through a node and this node is far

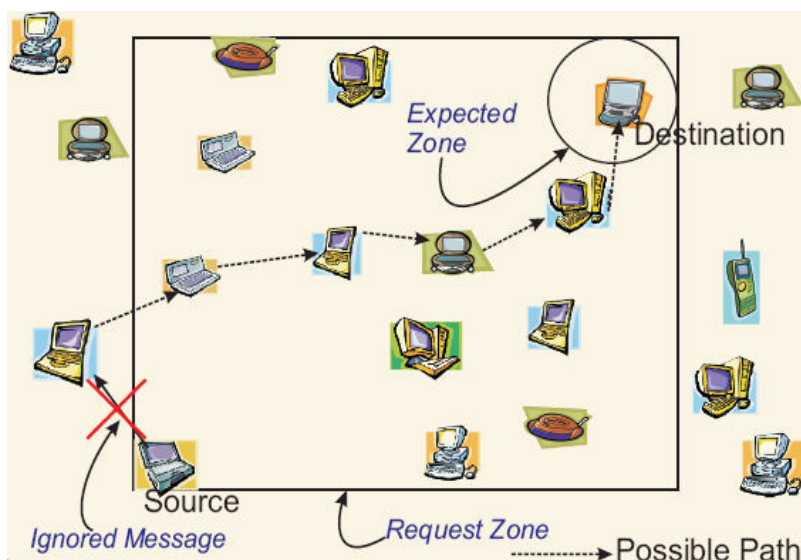


Figure 6.1: Delivery failure in LAR1 when a path it is available, problem detected by the methodology

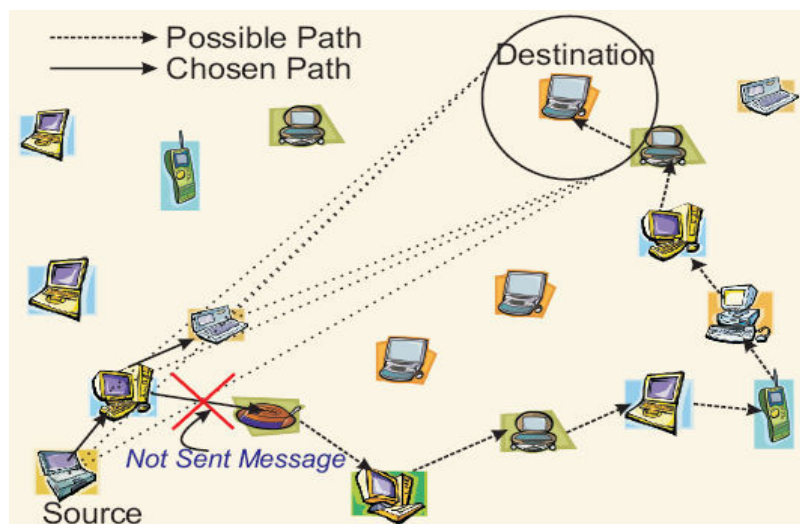


Figure 6.2: Delivery failure in LAR2, when a path it is available, problem detected by the methodology

from the destination than the previous one this path is discarded and the message will never reach the destination.

Figure 6.3 shows the loop scenario in LAR2. This situation occurs when all nodes are at the same distance from the destination, considering δ as a distance threshold. In this case the message will be forwarded from node to node indefinitely.

The loop on LAR1 occurs whenever the time during which a packet is retransmitted inside the requested zone is greater than the time nodes keep track of the transmitted packets.

The loop on DREAM, Figure 6.4, can occur if the search angle is greater than 90° . The path may not converge and loops are possible. This is somewhat unexpected as DREAM authors claim their algorithm being loop free [6]. This just reinforces the need of formal verification for routing protocols.

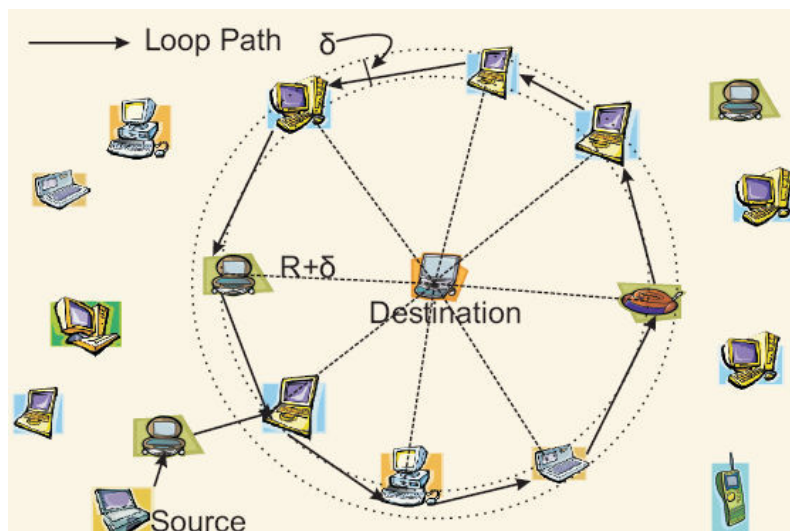


Figure 6.3: Loop detected in LAR2, when a path it is available, and all nodes are in the δ region in a near circle

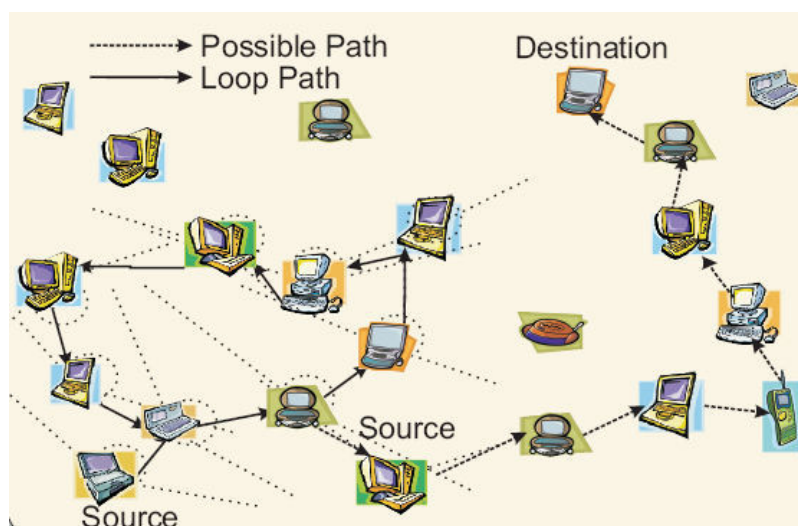


Figure 6.4: Loop and delivery failure discovered in DREAM when a series of nodes are in the angle of dissemination is big enough

Implementation	Depth reached	# States stored	# Transitions	Memory usage (MB) for states
DREAM	15	76	95	0.005
LAR	58	45529	52369	4.189
OLSR	37	61469	96637	3.442

Table 6.1: Sizes of the built models for the different algorithms

Chapter 7

Virtual Access Points For Vanets

The same method used to verify the routing protocols for wireless networks was also used to verify another kind of protocol, a cache like one. This protocol is specified for vehicular networks and intends to provide access to nodes in areas without access points. The work is also described in [62]. The verification of this protocol is important as it shows the potential of the technique. The previous works on this field focus mainly in routing protocols, which are important, but by far not the only kind of protocol and algorithm designed for wireless networks. The VAP protocol itself is not part of the contributions for this thesis, but we used our formal development technique in the development of the protocol, what greatly helped its development.

The network access on roads is normally provided by hot points, called Access Points (AP) installed in restaurants or gas stations. Such hot points even though providing a valuable service are not, even by far, enough to provide complete coverage to roads. Given the costs to cover the whole transport network one can expect that this situation will not change in the next few years. To make the access problem even worse the nodes in the network, the vehicles, are typically in high velocity, so the time node spend really connected, even in covered areas is small, this time is normally in the scale of few seconds. However, the need to be connected is increasing more and more, to the point that some people became dependent of this network connection. This chapter presents a simple, yet powerful, technique to provide coverage to nodes outside the hot spots areas.

Many solutions have being proposed to solve the roads uncovered problem, among them, the use of more powerful access points, satellite coverage, mobile APs [52], caches, among others. This proposal is related to the last one, caches. The first approaches, even though important and valid, have basically two problems that this work wants to avoid, scalability and cost. The cost to fill roads with access points, mainly mobile ones, is prohibitive. The satellite coverage, although important, simply does not scale for massive uplink/downlink communication. This approach, based on the cache solution, intends to provide a simple and inexpensive way to increase the coverage of VANETs.

The whole point of this work is to make nodes cooperate, in a coordinated way, to increase the network coverage area. Nodes that receive messages from the APs or from other nodes, act as a Virtual Access Points (VAP) to other nodes in specific non covered areas. In this way nodes, collaboratively, help to spread packets in areas that had previously no traffic. The technique is based in a cache approach, so the VAP can not guarantee that all packets will be delivered, it

is a best effort technique. Some of the applications that can heavily benefit from the technique introduced here are road information and tourist aid systems and stream based traffic.

A key aspect of our approach is how nodes cooperate, in a coordinated way, to increase the network coverage area. The mobile nodes cache messages originating from the APs, and are able to act as a VAP to other nodes in specific non covered areas. Thus, the nodes, collaboratively, help to forward packets to areas that had no traffic previously.

7.1 Related Work

The area of vehicular communication is a very prolific one. Many kinds of different techniques and target applications have been studied. This work focus in regular push based communication, where the user traffic demand is big, but, typically, not real time. We consider the basic multiple Infostation model introduced by Goodman et al. in [48] as the model to be followed here, however, we will refer to the Infostations as Access Points (APs).

The main objective of this work is to provide access to areas out of the coverage of APs. A very interesting work with similar objectives is the SPAWN system, introduced by Gerla et al. [45, 46]. In these, the authors discuss how vehicles should interact to accommodate swarming protocols, such as BitTorrent traffic. In SPAWN, the nodes passing through APs collect data that they subsequently exchange among nearby nodes. As opposed to our system, SPAWN focuses on a restricted application that generates great volumes of traffic. Nodes are required to carry possibly useless to them traffic and the BitTorrent protocol is bandwidth intensive. Moreover, the number of retransmissions of a message in a vehicular network is estimated to be approximately 3 and so our gain from using the swarming protocol in this environment is non-optimal.

The Data Mule project [47] and the Message Ferrying scheme [60], designed for sensor networks, propose the use of mobile nodes to collect data from the sensors, buffer it, and deliver the collected data to a sink. As opposed to these works, we consider the problem not of retrieving data from the nodes, but of disseminating it to them. The MULEs (Mobile Ubiquitous LAN Extensions) and ferries utilize nodes navigating through the sensor network to collect data in mobile caches. According to the Data Mule project, all the nodes are fixed and only the cache is mobile. On the contrast, in our scenario all nodes are mobile but we cannot affect their trajectories.

Message Ferrying also considers mobile nodes but in that approach, as well as in [50, 51], the nodes are required to follow specific paths and even move in order to help message delivering. The work presented in [51] proposes a multicast protocol for the highway environment where information dissemination though message flooding for VANET environments is proposed. Our proposal advocates that using of a more systematic approach for data dissemination, which is more bandwidth efficient.

Zebranet [61] uses Zebras to carry historical data to the sink. In this project latency is not important. However, the most important differences are that the network is extremely sparse, the nodes are not moving in roads, their speed is limited and when two nodes meet, we consider there is enough time to exchange larger volumes of information.

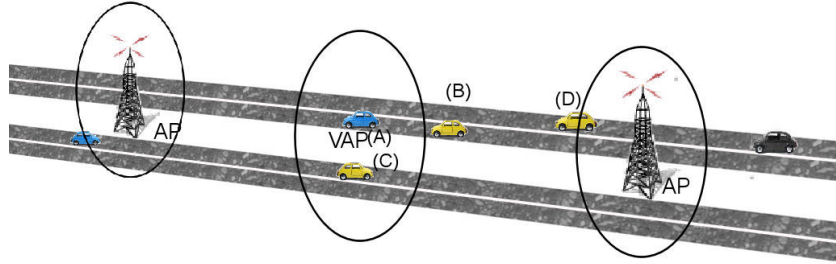


Figure 7.1: A road coverage vision

Chen et al. [49] study network delay as a function of the number of cars and their velocity. The authors note that node mobility on highways can improve end-to-end transmission delay when messages were relayed. Furthermore, that low density networks may experience higher delays. These results are directly related to our work. VAPs locations should be selected so that information is not too widely spread and messages of time sensitive applications can reach their destinations in time.

An appealing solution to the problem, but more expensive and probably hard to implement, is the system proposed by Gavrilovich in [52]. In this work, to compensate for the velocity of the cars, the authors propose the creation of a chain of mobile APs on the center of the highway. Their role is to increase coverage and compensate for the high velocity of cars on the highway.

7.2 The protocol

One of the biggest problems for mobile access is the lack of covered areas. The technique presented here intends to decrease the impact of such areas in a less intrusive way as possible. Basically the Virtual Access Point is a regular node, that when reach a predefined region, not covered by any AP, broadcasts the last messages received and stored into its buffer to the nearest nodes. As shown in Figure 7.1 the VAP increases the coverage to regions not previously covered by any regular APs. The main idea is that the mobile nodes receive the information they need independently of which node is providing it. We want the VAP to be, transparent for the nodes receiving the communication. The important thing is the information and not who is providing it, if a real AP or a VAP. In this way there is no difference in the downlink protocol from the AP and the VAP.

The regions where the nodes start to act as a VAP are chosen over the uncovered areas. The spots can be dynamically assigned, depending on the last AP position or fixed specified. The first one is more flexible, but the second one is easier to implement and is potentially closer to the optimal. If a node is passing through an area that is assigned as a VAP one, it listens the medium to see if there is no other node acting as a VAP in the region. If there is no one, the node starts to act as a VAP broadcasting the messages stored in its buffer. The mobile nodes, store the received messages in a buffer. The messages in this buffer are the ones shared if the mobile node decides to act as a VAP. The node act as a VAP just for a defined amount of time, the VAP is more valuable if the nodes that pass through the area act as VAP in rounds. This helps to keep the information fresher and it is fairer in relation to the use of nodes resources.

7.3 Protocol analysis

We assume that there is a finite set of objects, and that each time, each mobile node needs a subset of them. Such a subset can be the empty set or contain a number of objects, where each of them is associated with a deadline specified by the node. The deadline is the time after which the object is no longer needed and the query is dropped. When the node passes through roadside Access Points, it can query for objects and receive them with the answer, as well as overhear answers to queries of other nodes and objects that are proactively pushed by the AP. A cache is used to temporarily store objects for future queries. Whenever a node acts as VAP, it forms a program of cached objects to push to nearby vehicles.

The method works in a best-effort manner, there is no guarantee that the VAP will help nodes to receive all the messages they were meant to receive, but the VAPs are trying their best to accomplish it. In terms of exchanged messages, as expected, the VAP surely increase the number of exchanged messages among the nodes once now they are connected even in areas not covered by regular APs. The upper bound increase in the number of sent messages is given by:

$$nEM = nVAP * BS \quad (7.1)$$

where nEM is the number of exchanged messages, $nVAP$ is the number of Virtual Access points, and BS is the Buffer Size. Unfortunately not all received messages are useful for every node and the duplicate or old objects are discarded. The number and locations of the VAPs will greatly affect the systems performance. The data and latency of messages broadcasted on each a VAP depend on the distance from the originating APs.

The location of VAPs is also a sensitive parameter. A great concentration of VAPs near APs helps the dissemination of new information that the mobile nodes just received from the AP. Nodes moving towards an AP and passing through a VAP will receive messages that this AP has recently transmitted by nodes exiting the AP, and thus the effective range of this AP will be extended. However, in VAPs situated far from any AP, when the role of VAP is assumed by nodes that have not recently passed through any AP, the objects transmitted will be older, possibly less useful to receiving nodes. The further from APs a node is, the greater is the probability it will receive old and duplicated data, the points in between APs being the most likely candidates for such areas.

7.3.1 Methodology Application

We used our verification method to develop and evaluate the proposed technique. Our goal was to verify if the protocol is loop free. At first sight the protocol is loop free, however we discovered it may present loops. A loop scenario may occur when the relative velocities of the nodes are not equal. For example, considering the Figure 7.1, the simplest loop scenario occurs in the following case: node (A), acting as VAP in the point one, transmits the message M1 that is received by the node (B). Considering node (B) faster than node (A) and starting to act as a VAP at point two, it can transmit message M1, received by node (A). This characterizes a loop, and is one of the reasons why messages need to be equipped with unique IDs. Once the node (A) receives a duplicated message, identified by the ID, the node discards the message, indeed

preventing the loop formation.

Another kind of message loop is present, and in fact is even desirable. Again, consider Figure 7.1, take the node (A) acting as a VAP in the lane 1, a message M1 sent can reach the node (C), going in the opposite direction in the lane 2. At some point in the future the node (C) start to act as a VAP and retransmits the message M1 that is received by the node (D) in the lane 1. If node (D) does not have the message, it is stored and will be retransmitted in the future in case node (D) becomes a VAP. However notice that this case is not a loop in the conventional sense, once the nodes involved are different. Other point to observe is that this kind of loop is even desirable once it helps spreading messages over the region. The buffer favors newer messages, so older messages are ignored and removed from the buffer.

The protocols description allows for concurrent transmission. It is possible for two nodes to start acting as a VAP at the same time, when they are passing through the same region. This could become a problem since VAPs send messages using broadcast and the signal of the two VAPs will interfere with each other. For the simulation results we consider that the MAC layers mechanisms handle this, either using CSMA-CA protocol, like the one described for the IEEE 802.11 networks, or a scheduler, as it is the case for IEEE 802.16 networks. In any case the worst thing that will happen is a waist of bandwidth due concurrency or even collision.

As stated before, the technique is a best effort kind. There is no guarantee the mobile nodes will receive all the messages needed to fill their buffers, or even that they will receive any message at all. It may happen that a node traverses the entire path from one AP to the other without receiving any message from other VAPs. This may happen in case the node is unfortunate enough to not be inside the VAP range of other nodes acting as VAP, or when the node itself is acting the VAP, and thus is not receiving messages from other VAPs. These situations are more likely to occur in sparse networks.

7.3.2 Virtual Access Points for Stream Based Traffic Dissemination

We also extended the VAP protocol to handle stream based traffic and applied the VAP concept in a city environment. In this section we discuss how, and to what extent, VAPs can enable streaming on a highway or a city environment.

Figure 7.2 and Figure 7.3 demonstrates typical histograms of messages received in a 2Km simulated square of Washington DC and a highway segment respectively. Observe that VAPs provide a much more homogeneous and less intermittent distribution extending the areas where mobile nodes receive messages. The VAP technique was first designed to be used in road like environments, but as shown in Figure 7.2 metropolitan environments can too benefit from it.

7.3.2.1 Virtual Access Points for stream

The VAP protocol is basically the same one defined in the previous sections. The main focus of the VAP technique is to decrease the areas not covered by roadside APs so as to minimize the problem of intermittent access to mobile nodes. If we decrease this problem, then stream traffic for mobile users may be enabled.

The MAC control, again, is done by the lower layers, but when a node senses another node acting as VAP in the same region, it gives up being a VAP, even if it lies in the area it could

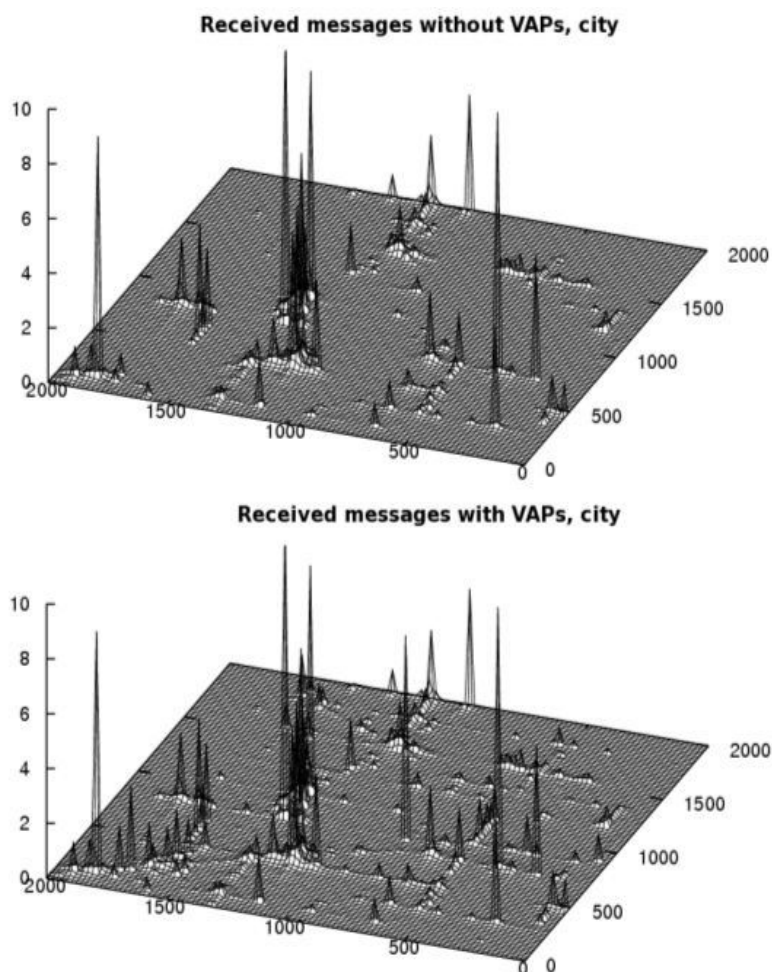


Figure 7.2: Typical receiving messages map for a 50 APs city scenario, we can see how VAPs allow us to connect existing "connectivity isles"

act as one. Therefore, the first node to broadcast VAP messages in a given region becomes the VAP. Nodes are not allowed to act as VAPs during two consecutive time intervals.

7.3.2.2 Analysis

The data stream is generated in a constant bit rate (CBR). At each second 1, 2 or 3 packets are generated from a source and spread through all antennas, which are then in charge of broadcasting the stream message to their neighbors. Each message is transmitted from each antenna just once. Every mobile node has a limited size buffer where it stores the last received messages. During cache replacement the oldest stream message, with lower stream ID, is discarded first, regardless if it was the last one to be received or not.

All the observations from the previous section holds in the case of streaming transmission. The system is a best effort one; there are no guarantees that every node will receive all stream packets, but using VAPs, we aim to increase the chances for timely reception. Note that the VAPs increase significantly the overall number of messages in the network; however, this increase occurs in areas with no previous coverage, so they create no significant interference with the normal network behavior. The number and locations of the VAPs greatly affect the system performance.

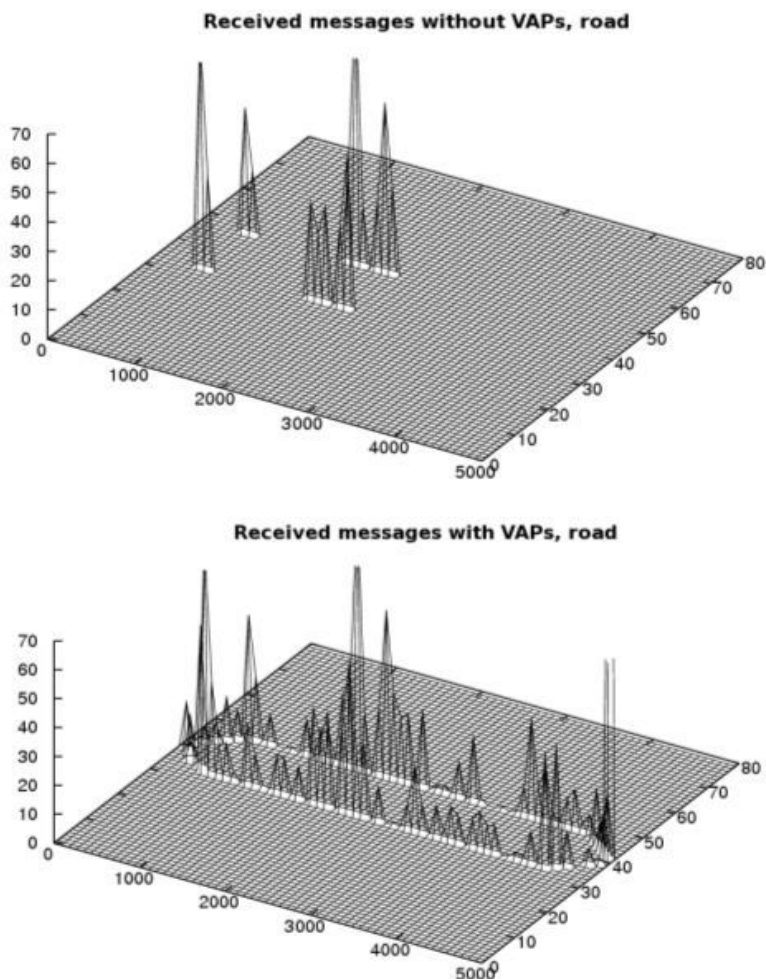


Figure 7.3: Typical receiving messages map for a 5 APs road scenario

Consequently, the role of VAP is assigned dynamically. Based on the nodes mobility pattern and distance from any APs, the nodes autonomously decide if they should act as a VAP.

7.3.3 Methodology Application

Interestingly the changes, in the model of the information source, stream vs implicit queries, had no impact in the model. All the previous cases of loops, lost messages and non-receiving messages continue to happen independently of the kind of traffic. However, we did change the way nodes perceive there are and VAPs around. This modifications on the model, ensured, if the lower levels work correctly, the inexistence of duplicated VAPs at the same area.

7.4 Remarks

This chapter introduced a simple yet powerful technique to increase network coverage for VENET networks. The technique is effective in increasing the packet reception rates with effectively no overhead on the AP regions. The formal verification of the technique provided useful insights of the protocol during its developing phase and indeed helped in the protocol development.

Chapter 8

Conclusions

Formal verification is a promising technique to validate algorithms for wireless networks. Differently of what someone can think the application of formal verification techniques in the development of new algorithms may be easy and still help increasing the quality of the in development protocols. The techniques presented here are a good start point for people who want to follow the research on this field or at least apply formal verification on their own algorithms.

The method presented here is robust and useful to confirm the existence of problems in algorithms and even to find new ones. With it, we were also able to verify flooding based protocols, which, in general, become a problem for other techniques because the state space explosion [16]. The topology abstraction, channel and information modeling showed to be an effective and reliable form to build verification models for routing algorithms.

However, the current version of the method does not help determining the protocol limits. Other problem is that the error scenarios must to be manually evaluated by the designer who also needs to determine the sources of such errors. Currently, we are working on extensions of this technique to other protocols and building a library of verified procedures that can be used to create more reliable protocols in the future. The protocol verification built on top of such procedures, may simplify the work of protocol designers and grant more reliable protocols in the future.

8.1 Directions for Future Research

The formal verification technique applied to algorithms for communication in wireless networks is a quite unexplored field yet, and therefore there are many opportunities for new research. Indeed, the field is in need of more specific techniques and tools.

Until this moment, at the best of our knowledge, no attempt was made trying to apply equivalence checking techniques in the verification of communication protocols for wireless networks. Equivalence checking is a powerful technique and may be extremely helpful in the development and mainly in the evolution of wireless routing algorithms.

Every new communication algorithm is a target for the techniques already developed. The verification of newer algorithms often reveals crucial failures that, if corrected earlier, can lead to more stable and trustable algorithms.

In terms of individual proposals, the work of Musuvathi et al. [40] has a huge merit in the

sense it presents a different and, in some terms, more practical approach. Verifying directly the algorithm code, instead models, may be an interesting and promising path to follow in the verification field. Advances in this kind of verification technique would have a wide applicability.

The work of Chiyangwa and Kwiatkowska [3] has also a remarkable value in the way it limits the verification scope and targets very specific limit problems. Such kind of approach may be interesting and applicable for other problems and situations. A good and valuable work, apart from expanding the existing one and applying it to other algorithms, could define a list of general situations and limits where one can use this kind of technique.

The mixing of the existent tools and approaches, for example the work of Obradovic et al. [2, 14], is also often interesting. One should use the techniques and tools that better suits its needs mix the use of tools using the right ones to prove the target characteristic is a valuable guideline.

Routing is a key aspect for the network and, mainly for wireless networks, security is a key aspect, and the work of Ács et al. [42] points this need. The verification of security aspects of routing protocols for wireless networks is also a promising research field.

Appendix A - PROMELA Codes

A.1 OLSR - Verification Code, first version

```
#define nnodes 3 /* Intermediates + Origin + Destination */
chan medium = [nnodes] of { mtype, byte, byte };
mtype = { Hello, TopologyControl, DataPacket };
mtype = { Origin, Destination, Intermediate };
bool packt = false;

proctype nodes() {
    bool ttl;
    byte type;
    byte node;

    /*build the packet*/
    if
        :: skip -> ttl = true /* reached the TTL*/
        :: skip -> ttl = false /* not reached the TTL*/
    fi;

    if
        :: skip -> type = Hello
        :: skip -> type = TopologyControl
        :: skip -> type = DataPacket
    fi;

    medium!type(ttl);

    if /*threat packet*/
        :: skip -> node = Origin /*acts as origin node*/
        :: skip -> node = Destination /*acts as origin node*/
        :: skip -> node = Intermediate /*acts as origin node*/
    fi;

    printf("###Pkt_type:%d ttl=%d ", type, ttl);
}

init {
    run nodes();
}
```

A.1.1 Spin Output - No error

hint: this search is more efficient if pan.c is compiled -DSAFETY
(Spin Version 4.2.9 -- 8 February 2007)
+ Partial Order Reduction

Bit statespace search for:

never claim	- (none specified)
assertion violations	+
acceptance cycles	- (not selected)
invalid end states	+

State-vector 32 byte, depth reached 7, errors: 0
34 states, stored
12 states, matched
46 transitions (= stored+matched)
0 atomic steps

hash factor: 246724 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):

0.002	equivalent memory usage for states (stored*(State-vector + overhead))
1.049	memory used for hash array (-w23)
0.040	memory used for bit stack
0.320	memory used for DFS stack (-m10000)
0.065	other (proc and chan stacks)
0.100	memory lost to fragmentation
1.573	total actual memory usage

unreached in proctype nodes
(0 of 25 states)

unreached in proctype :init:
(0 of 2 states)

A.2 OLSR - Verification Code, second version

```
#define nnodes 3 /* Intermediates + Origin + Destination */
mtype = { Hello,TopologyControl, DataPacket };
mtype = { Origin, Destination, Intermediate };
chan medium = [nnodes] of { mtype, byte, byte, bool };
byte origin;
byte destination;
byte ttl;          /* Msg time to live */
byte type;         /* Packet type */
byte node;

proctype network() {
    medium?type(origin,destination,ttl);

    if /*threat packet*/
        :: (true) -> node = Origin /*acts as origin node*/
        :: (true) -> node = Destination /*acts as origin node*/
        :: (true) -> node = Intermediate /*acts as origin node*/
    fi;
}

init {
    atomic {
        origin      = 0;
        destination = 2;
        if
            :: (true) -> ttl = true /* reached the TTL*/
            :: (true) -> ttl = false /* not reached the TTL*/

        fi;

        if
            :: (true) -> type = Hello
            :: (true) -> type = TopologyControl
            :: (true) -> type = DataPacket
        fi;

        medium!type(origin,destination,ttl) ->
        printf("-> Type: %d Origin: %d destination: %d ttl: %d\n",
            type, origin,destination,ttl);

    }/* atomic */
}
```

```
    run network()  
}
```

A.2.1 Spin Output - No error

hint: this search is more efficient if pan.c is compiled -DSAFETY
(Spin Version 4.2.9 -- 8 February 2007)
+ Partial Order Reduction

Bit statespace search for:

never claim	- (none specified)
assertion violations	+
acceptance cycles	- (not selected)
invalid end states	+

State-vector 36 byte, depth reached 10, errors: 0
55 states, stored
12 states, matched
67 transitions (= stored+matched)
9 atomic steps

hash factor: 152520 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):

0.003	equivalent memory usage for states (stored*(State-vector + overhead))
1.049	memory used for hash array (-w23)
0.040	memory used for bit stack
0.320	memory used for DFS stack (-m10000)
0.065	other (proc and chan stacks)
0.099	memory lost to fragmentation
1.573	total actual memory usage

unreached in proctype network
(0 of 10 states)

unreached in proctype :init:
(0 of 21 states)

A.3 OLSR - Verification Code, third version

```

/*===== Assertions to test out =====*/

#define T      (test==true)
#define p      (loop==true)
#define notp   (loop==false)
#define dm     (discardDataMsg == true)
#define dtc    (discardTCMsg == true)
#define ut     (updateTable == true)
#define unt    (updateNTable == true)
#define umpr   (updateMPR == true)
#define crt    (changedRoutTab == true)
#define cr     (calculateRoutes == true)
#define rr     (removeunusedRoutes == true)
#define rm     (resentMsg == true)
#define ct     (cleanTable == true)

/*===== Constants =====*/
#define true    1
#define false   0
#define retransmission 9 /* (nnodes * nnodes)*/
#define nnodes 3 /* Intermediates + Origin + Destination */

/*===== MESSAGE FORMAT =====*/
/* type, origin, destination, TTL */
chan medium = [nnodes] of { mtype, byte, byte, bool };

mtype = { Hello, TopologyControl, DataPacket };
mtype = { Origin, Intermediate, Destination };
bool packt = false;
bool test = true;

byte myid; /* choose in a randomicaly way the node id*/

/*===== Control Variables =====*/
byte origin_p; /* Origin of the message */
byte destination_p ; /* Destinatination of the message */
byte type; /* Packet type */
bool ttl; /* Time to Live of the message*/
bool knowsDestination; /* represents a search at the node rotting table */
bool newerTCinfo; /* If the info in the TC info is newer*/
bool tmp;

```

```
/*===== Log Variables =====*/
bool loop = false;
bool received = false;
bool discardDataMsg = false;
bool discardTCMsg = false;
bool updateTable = false;
bool updateNTable = false;
bool updateMPR = false;
bool changedRoutTab = false;
bool calculateRoutes = false;
bool removeunusedRoutes = false;
bool resentMsg = false;
bool cleanTable = false;

proctype network1() {
    /* Choosing randomly my ID. */
    if
        :: (true) -> myid = Origin
        :: (true) -> myid = Intermediate
        :: (true) -> myid = Destination
    fi;

    /*
     * receive the message
     */

    medium?type(origin_p,destination_p,ttl);

    /* DEBUG */
    printf("-> NET: Pkt_type:%d Origin:%d Destination:%d TTL:%d Myid:%d \n",
           type,origin_p,destination_p,ttl,myid);

end11:if /* I am the origin */
    :: (myid == Origin) -> loop = true ;

    /* I am the destination */
    :: (myid == Destination ) ->
end12:    if
        :: (type == DataPacket) -> received = true;
    fi;
```

```
        /* I am the intermiate node */
        :: (myid == Intermediate ) ->
end13:    if
        :: (type == Hello) ->
            updateNTable = true;
            updateMPR = true;
        :: (type == TopologyControl) ->
end14:    if
        :: (newerTCinfo == false) ->
            discardTCMsg= true;
        :: else ->
            cleanTable = true;
            updateTable = true;
        fi;
        :: (type == DataPacket) ->
end15:    if
        :: (knowsDestination == false) ->
            discardDataMsg = true;
        :: else ->
            resentMsg = true;
        fi;
        fi; /* Im the intermediate */
        fi; /* packet type */
    }

init {
    atomic {

        /* initialize node variables */

        if /* routing table hit/miss */
            :: (true) -> knowsDestination = true;
            :: (true) -> knowsDestination = false;
        fi->

        if /* TC message info newer or older */
            :: (true) -> newerTCinfo = true;
            :: (true) -> newerTCinfo = false;
        fi->

        if /*build the packet*/
```



```
:: packt == false -> packt = true->
/*
 * We have fixed the Origin and
 * the Destination to simplify
 * the model and its analysis.
 */
origin_p      = Origin;
destination_p = Destination;

if
  :: (true) -> ttl = true; /* reached the TTL */
  :: (true) -> ttl = false; /* not reached the TTL */
fi->

/*
 * Choosing a random value to the Packet Type
 */
if
  :: (true) -> type = Hello;
  :: (true) -> type = TopologyControl;
  :: (true) -> type = DataPacket;
fi;
fi; /* Build the packet */

/* put the message into the channel */
medium!type(origin_p,destination_p,ttl);

/* DEBUG */
printf("-> INIT: type:%d Origin:%d Destination:%d TTL:%d ",
       type,  origin_p,  destination_p,  ttl ) ->
} /*atomic*/

run network1();

}
```

A.3.1 Spin Output - No error

hint: this search is more efficient if pan.c is compiled -DSAFETY
(Spin Version 4.2.9 -- 8 February 2007)
+ Partial Order Reduction

Bit statespace search for:

```
never claim           - (none specified)
assertion violations  +
acceptance cycles    - (not selected)
invalid end states    +
```

State-vector 36 byte, depth reached 18, errors: 0
637 states, stored
0 states, matched
637 transitions (= stored+matched)
42 atomic steps

hash factor: 13168.9 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):

```
0.033  equivalent memory usage for states
(stored*(State-vector + overhead))
1.049  memory used for hash array (-w23)
0.040  memory used for bit stack
0.320  memory used for DFS stack (-m10000)
0.070  other (proc and chan stacks)
0.095  memory lost to fragmentation
1.573  total actual memory usage
```

unreached in proctype network1
(0 of 41 states)

unreached in proctype :init:
(0 of 37 states)

A.3.2 Spin Output - Error found, Supertrace Mode

warning: for p.o. reduction to be valid the never claim must
be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 5 (line 175)
pan: claim violated! (at depth 28)

```
pan: wrote pan_in1.trail
(Spin Version 4.2.9 -- 8 February 2007)
Warning: Search not completed
+ Partial Order Reduction

Bit statespace search for:
never claim          +
assertion violations + (if within scope of claim)
acceptance cycles   + (fairness disabled)
invalid end states  - (disabled by never claim)

State-vector 40 byte, depth reached 32, errors: 1
  389 states, stored
  43 states, matched
  432 transitions (= stored+matched)
  28 atomic steps

hash factor: 172516 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
0.022 equivalent memory usage for states
(stored*(State-vector + overhead))
8.389 memory used for hash array (-w26)
0.400 memory used for bit stack
3.200 memory used for DFS stack (-m100000)
0.111 other (proc and chan stacks)
0.094 memory lost to fragmentation
12.194 total actual memory usage

unreached in proctype network1
(0 of 41 states)
unreached in proctype :init:
(0 of 37 states)
unreached in proctype :never:
line 180, "pan.____", state 8, "-end-"
(1 of 8 states)
```

A.3.3 Trace - LTL : `!!dm`

```

preparing trail, please wait...done
Starting :init: with pid 0
spin: warning, "pan_in", global, 'bit    test' variable is never used
spin: warning, "pan_in", global, 'bit    tmp' variable is never used
spin: warning, "pan_in", global, 'bit    loop' variable is never used
spin: warning, "pan_in", global, 'bit    received' variable is never used
spin: warning, "pan_in", global, 'bit    discardDataMsg' variable is never used
spin: warning, "pan_in", global, 'bit    discardTCMsg' variable is never used
spin: warning, "pan_in", global, 'bit    updateTable' variable is never used
spin: warning, "pan_in", global, 'bit    updateNTable' variable is never used
spin: warning, "pan_in", global, 'bit    updateMPR' variable is never used
spin: warning, "pan_in", global, 'bit    changedRoutTab' variable is never used
spin: warning, "pan_in", global, 'bit    calculateRoutes' variable is never used
spin: warning, "pan_in", global, 'bit    removeunusedRoutes' variable is never
used
spin: warning, "pan_in", global, 'bit    resentMsg' variable is never used
spin: warning, "pan_in", global, 'bit    cleanTable' variable is never used
spin: couldn't find claim (ignored)
  2: proc  0 (:init:) line 118 "pan_in" (state 3) [(1)] <
  2: proc  0 (:init:) line 118 "pan_in" (state 4) [knowsDestination = 0]
  3: proc  0 (:init:) line 122 "pan_in" (state 7) [(1)] <
  3: proc  0 (:init:) line 122 "pan_in" (state 8) [newerTCinfo = 1]
  4: proc  0 (:init:) line 127 "pan_in" (state 13) [((packt==0))] <
  4: proc  0 (:init:) line 127 "pan_in" (state 14) [packt = 1] <
  4: proc  0 (:init:) line 133 "pan_in" (state 15) [origin_p = Origin] <

  4: proc  0 (:init:) line 134 "pan_in" (state 16)
    [destination_p = Destination] <
  5: proc  0 (:init:) line 137 "pan_in" (state 17) [(1)] <
  5: proc  0 (:init:) line 137 "pan_in" (state 18) [ttl = 1]
  6: proc  0 (:init:) line 147 "pan_in" (state 27) [(1)] <
  6: proc  0 (:init:) line 147 "pan_in" (state 28) [type = DataPacket]
  7: proc  0 (:init:) line 152 "pan_in" (state -) [values: 1!DataPacket,6,4,1]
  7: proc  0 (:init:) line 152 "pan_in" (state 33)
    [medium!type,origin_p,destination_p,ttl] <
-> INIT: type:1 Origin:6 Destination:4 TTL:1  7:
proc  0 (:init:) line 155 "pan_in" (state 34)
[printf('-> INIT: type:%d Origin:%d Destination:%d TTL:%d ',
type,origin_p,destination_p,ttl)]
Starting network1 with pid 2
  9: proc  0 (:init:) line 159 "pan_in" (state 36) [(run network1())]

```

```
11: proc 1 (network1) line 61 "pan_in" (state 3) [(1)]
13: proc 1 (network1) line 61 "pan_in" (state 4) [myid = Intermediate]
15: proc 1 (network1) line 69 "pan_in" (state -) [values: 1?DataPacket,6,4,1]
15: proc 1 (network1) line 69 "pan_in" (state 9)
[medium?type,origin_p,destination_p,ttl]
-> NET: Pkt_type:1 Origin:6 Destination:4 TTL:1 Myid:5
17: proc 1 (network1) line 72 "pan_in" (state 10)
[printf('-> NET: Pkt_type:%d Origin:%d Destination:%d TTL:%d Myid:%d
\\n',type,origin_p,destination_p,ttl,myid)]
19: proc 1 (network1) line 86 "pan_in" (state 18) [((myid==Intermediate))]
21: proc 1 (network1) line 99 "pan_in" (state 30) [((type==DataPacket))]
23: proc 1 (network1) line 101 "pan_in" (state 31) [((knowsDestination==0))]
25: proc 1 (network1) line 102 "pan_in" (state 32) [discardDataMsg = 1]
27: proc 1 terminates
29: stop error, proc 1 (i=2) trans 40, @
#processes: 1
29: proc 0 (:init:) line 161 "pan_in" (state 37)
2 processes created
```

A.4 OLSR - Verification Code, two paralel messages version

```

/*===== Assertions to test out =====*/

#define T      (test==true)
#define p      (loop==true)
#define notp   (loop==false)
#define dm     (discardDataMsg == true)
#define dtc    (discardTCMsg == true)
#define ut     (updateTable == true)
#define unt    (updateNTable == true)
#define umpr   (updateMPR == true)
#define crt    (changedRoutTab == true)
#define cr     (calculateRoutes == true)
#define rr     (removeunusedRoutes == true)
#define rm     (resentMsg == true)
#define ct     (cleanTable == true)

/*===== Constants =====*/
#define true    1
#define false   0
#define retransmission 9 /* (nnodes * nnodes)*/
#define nnodes 3 /* Intermediates + Origin + Destination */

/*===== MESSAGE FORMAT =====*/
/* type, origin, destination, TTL */
chan medium = [nnodes] of { mtype, byte, byte, bool };

mtype = { Hello, TopologyControl, DataPacket };
mtype = { Origin, Intermediate, Destination };
bool packt = false;
bool test = true;

byte myid; /* choose in a randomicaly way the node id*/

/*===== Control Variables =====*/
byte origin_p; /* Origin of the message */
byte destination_p ; /* Destinatination of the message */
byte type; /* Packet type */
bool ttl; /* Time to Live of the message*/
bool knowsDestination; /* represents a search at the node rotting table */
bool newerTCinfo; /* If the info in the TC info is newer */
bool tmp;

```

```
/*===== Log Variables =====*/
bool loop = false;
bool received = false;
bool discardDataMsg = false;
bool discardTCMsg = false;
bool updateTable = false;
bool updateNTable = false;
bool updateMPR = false;
bool changedRoutTab = false;
bool calculateRoutes = false;
bool removeunusedRoutes = false;
bool resentMsg = false;
bool cleanTable = false;

/*****
 * Models the node's behavior
 * In this case with two variables it
 * models the behavior of a node when
 * handling two concurrent transmissions
 *****/
proctype network1() {
    byte  type1;    /* Packet type */
    bool  ttl1;    /* Time to Live of the message*/

    /* Choosing randomly my ID. */
    if

        :: (true) -> myid = Origin
        :: (true) -> myid = Intermediate
        :: (true) -> myid = Destination
    fi;

    /*
     * receive the message
     */
    medium?type1(origin_p,destination_p,ttl1);

    /*  DEBUG  */
    printf("-> NET: Pkt_type:%d Origin:%d Destination:%d TTL:%d Myid:%d \n",
           type1,origin_p,destination_p,ttl1,myid);
}
```

```

end11:if /* I am the origin */
    :: (myid == Origin) -> loop = true ;

    /* I am the destination */
    :: (myid == Destination ) ->
end12:    if
        :: (type1 == DataPacket) -> received = true;
        fi;

    /* I am the intermiate node */
    :: (myid == Intermediate ) ->
end13:    if
        :: (type1 == Hello) ->
            updateNTable = true;
            updateMPR = true;
        :: (type1 == TopologyControl) ->
end14:    if
        :: (newerTCinfo == false) ->
            discardTCMsg= true;
        :: else ->
            cleanTable = true;
            tmp = knowsDestination;
            knowsDestination = false;
            updateTable = true;
            knowsDestination =tmp;
        fi;
        :: (type1 == DataPacket) ->
end15:    if
        :: (knowsDestination == false) ->
            discardDataMsg = true;
        :: else ->
            resentMsg = true;
        fi;
        fi; /* Im the intermediate */
    fi; /* packet type */
}

```

```

/*****
 * Initialization Process Should be
 * as random and as broad as possible
 *****/
init {

```



```
atomic {

    /* initialize node variables */

    if /* routing table hit/miss */
        :: (true) -> knowsDestination = true;
        :: (true) -> knowsDestination = false;
    fi->

    if /* TC message info newer or older */
        :: (true) -> newerTCinfo = true;
        :: (true) -> newerTCinfo = false;
    fi->

    /*
     * We have fixed the Origin and
     * the Destination to simplify
     * the model and its analysis.
     */
    origin_p      = Origin;
    destination_p = Destination;

    if
        :: (true) -> ttl = true; /* reached the TTL */
        :: (true) -> ttl = false; /* not reached the TTL */
    fi->

    /*
     * Choosing a random value to the Packet Type
     */
    if
        :: (true) -> type = Hello;
        :: (true) -> type = TopologyControl;
        :: (true) -> type = DataPacket;
    fi;

    /* put the message into the channel */
    medium!type(origin_p,destination_p,ttl);

    /* DEBUG */
    printf("-> INIT: type:%d Origin:%d Destination:%d TTL:%d ",
           type,  origin_p,  destination_p,  ttl );
}
```

```
if
    :: (true) -> ttl = true; /* reached the TTL */
    :: (true) -> ttl = false; /* not reached the TTL */
fi->

/*
 * Choosing a random value to the Packet Type
 */
if
    :: (true) -> type = Hello;
    :: (true) -> type = TopologyControl;
    :: (true) -> type = DataPacket;
fi;

/* put the message into the channel */
medium!type(origin_p,destination_p,ttl);

/* DEBUG */
printf("-> INIT: type:%d Origin:%d Destination:%d TTL:%d ",
        type, origin_p, destination_p, ttl ) ->
}

run network1(); run network1();
}
```

A.4.1 Spin Output - No error

hint: this search is more efficient if pan.c is compiled -DSAFETY
 (Spin Version 4.2.9 -- 8 February 2007)
 + Partial Order Reduction

Bit statespace search for:

```

never claim          - (none specified)

assertion violations +
acceptance cycles   - (not selected)
invalid end states   +

```

State-vector 40 byte, depth reached 37, errors: 0
 61469 states, stored
 35168 states, matched
 96637 transitions (= stored+matched)
 254 atomic steps

hash factor: 136.469 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):

```

3.442  equivalent memory usage for states (stored*(State-vector +
overhead))
1.049  memory used for hash array (-w23)
0.040  memory used for bit stack
0.320  memory used for DFS stack (-m10000)
0.174  other (proc and chan stacks)
0.094  memory lost to fragmentation
1.676  total actual memory usage

```

unreached in proctype network1
 (0 of 44 states)

unreached in proctype :init:
 (0 of 50 states)

A.4.2 Spin Output - Error found, Supertrace Mode

warning: for p.o. reduction to be valid the never claim must be
 stutter-invariant
 (never claims generated from LTL formulae are stutter-invariant)
 depth 0: Claim reached state 5 (line 197)

```
pan: claim violated! (at depth 58)
pan: wrote pan_in1.trail
(Spin Version 4.2.9 -- 8 February 2007)
Warning: Search not completed
+ Partial Order Reduction
```

```
Bit statespace search for:
never claim          +
assertion violations + (if within scope of claim)
acceptance  cycles  + (fairness disabled)
invalid end states - (disabled by never claim)
```

```
State-vector 44 byte, depth reached 68, errors: 1
  3999 states, stored
  2091 states, matched
  6090 transitions (= stored+matched)
  19 atomic steps
```

```
hash factor: 16781.4 (best if > 100.)
```

```
bits set per state: 3 (-k3)
```

```
Stats on memory usage (in Megabytes):
```

```
0.240 equivalent memory usage for states (stored*(State-vector + overhead))
8.389 memory used for hash array (-w26)
0.400 memory used for bit stack
3.200 memory used for DFS stack (-m100000)
0.112 other (proc and chan stacks)
0.093 memory lost to fragmentation
12.194 total actual memory usage
```

```
unreached in proctype network1
line 96, "pan.____", state 24, "discardTCMsg = 1"
(1 of 44 states)
unreached in proctype :init:
line 126, "pan.____", state 11, "(1)"
line 126, "pan.____", state 11, "(1)"
(1 of 50 states)
unreached in proctype :never:
line 202, "pan.____", state 8, "-end-"
(1 of 8 states)
```

A.4.3 Trace - *LTL*: `!dm`, distinct case

```

preparing trail, please wait...done
Starting :init: with pid 0
spin: warning, "pan_in", global, 'bit   packt' variable is never used
spin: warning, "pan_in", global, 'bit   test' variable is never used
spin: warning, "pan_in", global, 'bit   loop' variable is never used
spin: warning, "pan_in", global, 'bit   received' variable is never used
spin: warning, "pan_in", global, 'bit   discardDataMsg' variable is never used
spin: warning, "pan_in", global, 'bit   discardTCMsg' variable is never used
spin: warning, "pan_in", global, 'bit   updateTable' variable is never used
spin: warning, "pan_in", global, 'bit   updateNTable' variable is never used
spin: warning, "pan_in", global, 'bit   updateMPR' variable is never used
spin: warning, "pan_in", global, 'bit   changedRoutTab' variable is never used
spin: warning, "pan_in", global, 'bit   calculateRoutes' variable is never used
spin: warning, "pan_in", global, 'bit   removeunusedRoutes' variable is never
used
spin: warning, "pan_in", global, 'bit   resentMsg' variable is never used
spin: warning, "pan_in", global, 'bit   cleanTable' variable is never used
spin: couldn't find claim (ignored)
  2: proc  0 (:init:) line 122 "pan_in" (state 1) [(1)] <
  2: proc  0 (:init:) line 122 "pan_in" (state 2) [knowsDestination = 1]
  3: proc  0 (:init:) line 127 "pan_in" (state 7) [(1)] <
  3: proc  0 (:init:) line 127 "pan_in" (state 8) [newerTCinfo = 1] <
  3: proc  0 (:init:) line 134 "pan_in" (state 13) [origin_p = Origin] <
  3: proc  0 (:init:) line 135 "pan_in" (state 14) [destination_p =
Destination] <

  4: proc  0 (:init:) line 138 "pan_in" (state 15) [(1)] <
  4: proc  0 (:init:) line 138 "pan_in" (state 16) [ttl = 1]
  5: proc  0 (:init:) line 147 "pan_in" (state 23) [(1)] <
  5: proc  0 (:init:) line 147 "pan_in" (state 24) [type = TopologyControl]
  6: proc  0 (:init:) line 152 "pan_in" (state -) [values:
1!TopologyControl,6,4,1]
  6: proc  0 (:init:) line 152 "pan_in" (state 29)
  [medium!type,origin_p,destination_p,ttl] <
-> INIT: type:2 Origin:6 Destination:4 TTL:1   6: proc
0 (:init:) line 155 "pan_in" (state 30) [printf('->
INIT: type:%d Origin:%d Destination:%d TTL:%d ',type,origin_p,
destination_p,ttl)]
  7: proc  0 (:init:) line 160 "pan_in" (state 31) [(1)] <
  7: proc  0 (:init:) line 160 "pan_in" (state 32) [ttl = 1]
  8: proc  0 (:init:) line 170 "pan_in" (state 41) [(1)] <

```

```

8: proc 0 (:init:) line 170 "pan_in" (state 42) [type = DataPacket]
9: proc 0 (:init:) line 174 "pan_in" (state -) [values:
1!DataPacket,6,4,1]
9: proc 0 (:init:) line 174 "pan_in" (state 45)
[medium!type,origin_p,destination_p,ttl] <
-> INIT: type:1 Origin:6 Destination:4 TTL:1 9: proc 0
(:init:) line 177 "pan_in" (state 46) [printf('-> INIT:
type:%d Origin:%d Destination:%d TTL:%d ',
type,origin_p,destination_p,ttl)]
Starting network1 with pid 2

11: proc 0 (:init:) line 181 "pan_in" (state 48) [(run network1())]
13: proc 1 (network1) line 63 "pan_in" (state 1) [(1)]
Starting network1 with pid 3
15: proc 0 (:init:) line 181 "pan_in" (state 49) [(run network1())]
17: proc 2 (network1) line 64 "pan_in" (state 3) [(1)]
19: proc 1 (network1) line 63 "pan_in" (state 2) [myid = Origin]
21: proc 2 (network1) line 64 "pan_in" (state 4) [myid = Intermediate]
23: proc 2 (network1) line 72 "pan_in" (state -)
[values: 1?TopologyControl,6,4,1]
23: proc 2 (network1) line 72 "pan_in" (state 9)
[medium?type1,origin_p,destination_p,ttl1]
-> NET: Pkt_type:2 Origin:6 Destination:4 TTL:1 Myid:5
25: proc 2 (network1) line 75 "pan_in" (state 10) [printf('->
NET: Pkt_type:%d Origin:%d Destination:%d TTL:%d Myid:%d \\n',
type1,origin_p,destination_p,ttl1,myid)]
27: proc 2 (network1) line 88 "pan_in" (state 18) [((myid==Intermediate))]
29: proc 2 (network1) line 93 "pan_in" (state 22)
[((type1==TopologyControl))]
31: proc 2 (network1) line 97 "pan_in" (state 25) [else]
33: proc 2 (network1) line 98 "pan_in" (state 26) [cleanTable = 1]
35: proc 2 (network1) line 99 "pan_in" (state 27) [tmp = knowsDestination]
37: proc 2 (network1) line 100 "pan_in" (state 28) [knowsDestination = 0]
39: proc 2 (network1) line 101 "pan_in" (state 29) [updateTable = 1]
41: proc 1 (network1) line 72 "pan_in" (state -)
[values: 1?DataPacket,6,4,1]
41: proc 1 (network1) line 72 "pan_in" (state 9)
[medium?type1,origin_p,destination_p,ttl1]
-> NET: Pkt_type:1 Origin:6 Destination:4 TTL:1 Myid:5
43: proc 1 (network1) line 75 "pan_in" (state 10) [printf('->
NET: Pkt_type:%d Origin:%d Destination:%d TTL:%d Myid:%d \\n',
type1,origin_p,destination_p,ttl1,myid)]

```

```
45: proc 1 (network1) line 88 "pan_in" (state 18) [((myid==Intermediate))]
47: proc 1 (network1) line 104 "pan_in" (state 33) [((type1==DataPacket))]
49: proc 1 (network1) line 106 "pan_in" (state 34) [((knowsDestination==0))]
51: proc 2 (network1) line 102 "pan_in" (state 30) [knowsDestination = tmp]
53: proc 2 terminates
55: proc 1 (network1) line 107 "pan_in" (state 35) [discardDataMsg = 1]
57: proc 1 terminates
59: proc 0 terminates
spin: trail ends after 59 steps
3 processes created
```

A.5 OLSR - Last version

```

/*===== Assertions to test out =====*/

#define T      (test==true)
#define p      (loop==true)
#define notp   (loop==false)
#define dm     (discardDataMsg == true)
#define dtc    (discardTCMsg == true)
#define ut     (updateTable == true)
#define unt    (updateNTable == true)
#define umpr   (updateMPR == true)
#define crt    (changedRoutTab == true)
#define cr     (calculateRoutes == true)
#define rr     (removeunusedRoutes == true)
#define rm     (resentMsg == true)
#define ct     (cleanTable == true)
#define receivedMTO (receivedMoreThanOne == true)
#define NotDellivery (reachDestination == false)
#define Forwarded (forward == true)
#define loop_teste (loop == true)

/*===== Constants =====*/
#define true    1
#define false   0
#define retransmission  4
#define nnodes  3 /* Intermediates + Origin + Destination */

/*===== MESSAGE FORMAT =====*/
/* type,  origin, destination, TTL, last id,
   number of retransmissions*/
chan medium = [nnodes] of { mtype, byte,  byte,          bool, byte,
                           byte };

mtype = { Hello, TopologyControl, DataPacket };
mtype = { Origin, Intermediate, Destination };
bool packt = false;
bool test = true;

byte myid;          /* choose in a randomicaly way the node id*/

/*===== Control Variables =====*/

```



```

byte  origin_p;      /* Origin of the message */
byte  destination_p ; /* Destinatation of the message */
byte  type;          /* Packet type */
bool  ttl;           /* Time to Live of the message*/
bool  knowsDestination; /* represents a search at the node roting table */
bool  newerTCinfo;   /* If the info in the TC info is newer*/
bool  tmp;

bool  again;         /* This Flag is used to do the choosing of sending once
                    or twice the message */

byte  ret_control;

/*===== Log Variables =====*/
bool  loop = false;
bool  data_loop = false;
bool  received = false;
bool  discardDataMsg = false;
bool  discardTCMsg = false;
bool  updateTable = false;
bool  updateNTable = false;
bool  updateMPR = false;
bool  changedRoutTab = false;
bool  calculateRoutes = false;
bool  removeunusedRoutes = false;
bool  resentMsg = false;
bool  cleanTable = false;
bool  reachDestination = false;
bool  receivedMoreThanOne = false;
bool  forward = false;
bool  loop_int = false;

/*****
 * Models the node's behavior
 * In this case with two variables it
 * models the behavior of a node when
 * handling two concurrent transmissions
 *****/
proctype network1() {
    byte  type1;      /* Packet type */
    bool  ttl1;       /* Time to Live of the message*/

```

```
byte priorid_p;          /* choose in a randomicaly way the node id*/
byte retransm_p;        /* Number of retransmissions*/

/* Choosing randomly my ID. */
if
  :: (true) -> myid = Origin
  :: (true) -> myid = Intermediate
  :: (true) -> myid = Destination
fi;

if /* routing table hit/miss */
  :: (true) -> knowsDestination = true;
  :: (true) -> knowsDestination = false;
fi->

/*
 * Choosing a random value to "again" which, in a reality, chooses
 * if it sends 1 ou 2 copies of the message. When two copies
 * are sent it is done to emulate the broadcasting by choosing "again ==true"
 */

if
  :: (true) -> again = true
  :: (true) -> again = false
fi;

/*
 * receive the message
 */
medium?type1(origin_p,destination_p,ttl1,priorid_p,retransm_p);

/* DEBUG */
printf("-> NET: Pkt_type:%d Origin:%d Destination:%d TTL:%d Myid:%d
      priorNode:%d retransm:%d \n",
      type1,origin_p,destination_p,ttl1,myid,priorid_p,retransm_p);

/*
 * If the node is in the communication range
 */
```

```
end10:if
    :: (retransm_p < retransmission) ->

end11:if /* I am the origin */
    :: (myid == Origin) ->

    /* loop detection at the source. Ignore message I sent to myself */
end112:    if
            :: (priorid_p != 0) -> loop = true
        fi

        /* I am the destination */
        :: (myid == Destination ) ->

end113:        if
                :: (reachDestination == true) -> receivedMoreThanOne = true
                :: (reachDestination == false) -> reachDestination = true
            fi ->

end12:        if
                :: (type1 == DataPacket) -> received = true;
            fi;

        /* I am the intermiante node */
        :: (myid == Intermediate ) ->
end13:        if
                :: (type1 == Hello) ->
                    updateNTable = true;
                    updateMPR = true;
                :: (type1 == TopologyControl) ->
end14:        if
                :: (newerTCinfo == false) ->
                    discardTCMsg= true;
                :: else ->
                    cleanTable = true;
                    tmp = knowsDestination;
                    knowsDestination = false;
                    updateTable = true;
                    knowsDestination =tmp;
            fi;
                :: (type1 == DataPacket) ->
end15:        if
```

```

        :: (knowsDestination == false) ->
            discardDataMsg = true;
        :: else ->
            resentMsg = true;
    fi;

    /* TTL haven't expired. So, we can forward the message */
end151:    if
        :: (ttl1 == false && type1 != Hello ) ->
end152:    if
        :: (forward == true) -> loop_int = true
        :: (forward == false) -> forward = true ->
            medium!type1(origin_p,destination_p,ttl1,myid,
                (retransm_p + 1));
        /*
        * Sending the message again or not
        */
end153:    if
        :: (again == true) ->
            medium!type1(origin_p,destination_p,ttl1,myid,
                (retransm_p + 1));
        :: (again == false) -> again = false /* doing nothing */
    fi ->

        run network1();

    fi;
    /* TTL have expired, or is Hello. We need ignore the message */
    /* :: else -> ; */
    fi;

    fi; /* Im the intermediate */
    fi; /* packet type */
    fi; /* in the range */
}

/*****
* Initialization Process Should be
* as random and as broad as possible
*****/
init {

```

```
atomic {

    ret_control = 0;

    /* initialize node variables */

    if /* TC message info newer or older */
        :: (true) -> newerTCinfo = true;
        :: (true) -> newerTCinfo = false;
    fi->

    /*
     * We have fixed the Origin and
     * the Destination to simplify
     * the model and its analysis.
     */
    origin_p      = Origin;
    destination_p = Destination;

    if
        :: (true) -> ttl = true; /* reached the TTL */
        :: (true) -> ttl = false; /* not reached the TTL */
    fi->

    /*
     * Choosing a random value to the Packet Type
     */
    if
        :: (true) -> type = Hello;
        :: (true) -> type = TopologyControl;
        :: (true) -> type = DataPacket;
    fi;

    /*
     * Choosing a random the prior node
     */
    byte prior_p;
    if
        :: (true) -> prior_p = Origin;
        :: (true) -> prior_p = Intermediate;
    fi;
}
```

```
/* put the message into the channel */
medium!type(origin_p,destination_p,ttl, prior_p,0);

/* DEBUG */
printf("-> INIT: type:%d Origin:%d Destination:%d TTL:%d, prior=%d ",
        type,  origin_p,  destination_p,  ttl, prior_p );

if
  :: (true) -> ttl = true; /* reached the TTL */
  :: (true) -> ttl = false; /* not reached the TTL */
fi->

/*
 * Choosing a random value to the Packet Type
 */
if
  :: (true) -> type = Hello;
  :: (true) -> type = TopologyControl;
  :: (true) -> type = DataPacket;
fi;

/*
 * Choosing a random the prior node
 */
if
  :: (true) -> prior_p = Origin;
  :: (true) -> prior_p = Intermediate;
fi;

/* put the message into the channel */
medium!type(origin_p,destination_p,ttl, prior_p, 0);

/* DEBUG */
printf("-> INIT: type:%d Origin:%d Destination:%d TTL:%d, prior=%d ",
        type,  origin_p,  destination_p,  ttl, prior_p )->
}

run network1();  run network1();
}
```

Spin Output - No error

hint: this search is more efficient if pan.c is compiled -DSAFETY

```
Depth=      69 States=      1e+06 Transitions= 1.59e+06 Memory=      1.501
t=    2.17 R=    5e+05
Depth=      89 States=      2e+06 Transitions= 3.15e+06 Memory=      1.501
t=     4.6 R=    4e+05
```

(Spin Version 5.2.0 -- 2 May 2009)

+ Partial Order Reduction

Bit statespace search for:

```
never claim          - (none specified)
assertion violations +
acceptance cycles   - (not selected)
invalid end states   +
```

State-vector 76 byte, depth reached 89, errors: 0

2245174 states, stored

1288078 states, matched

3533252 transitions (= stored+matched)

547 atomic steps

hash factor: 3.73628 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):

196.987 equivalent memory usage for states (stored*(State-vector + overhead))

1.000 memory used for hash array (-w23)

0.038 memory used for bit stack

0.305 memory used for DFS stack (-m10000)

1.501 total actual memory usage

unreached in proctype network1

(0 of 85 states)

unreached in proctype :init:

(0 of 57 states)

A.6 DREAM Verification Code

```

/*===== Assertions to test out =====*/

#define p (loop==true)
#define notp (loop==false)
#define q (nnodes!=3)
#define t (trash == 1)
#define sa (sendAck == true)
#define dm (duplicatedMessage == true)
#define ip (IgnoredPacket == true)
#define ipna ((IgnoredPacket == true) && (NoNeighborInArea == true))

/*=====*/

#define true 1
#define false 0
#define TTL 9
#define range 9 /* (nnodes * nnodes)*/
#define nnodes 3 /* Intermediates + Origin + Destination */

/* ===== */

/* ===== MESSAGE FORMAT + CHAIN ===== */

/* type, origin, destination, TTL , Angle */
chan medium = [nnodes] of {mtype, byte, byte, byte, byte };

/* ===== GLOBAL VARIABLES ===== */

/* packet types */
mtype = {AckPacket, DataPacket};

byte myid; /* choose in a randomicaly way the node id*/

byte origin; /* Origin of the message */

```



```
byte ttl;          /* Time to Live of the message*/
byte destination ; /* Destinatination of the message */

byte destinationTimestamp; /* the age of the destination information */
byte angle ;       /* the angle (alpha) in the region to the destination */

bool again;        /* This Flag is used to do the choosing of sending once
                    or twice the message */
byte previousAngle; /* The flooding angle of the previous node */

bool sentAck = false ; /* This flag indicate if the ack was sent or not */

bool duplicatedMessage = false; /* This flag indicate if was sent a
                                duplicated message */

bool HaveDestPosition; /* indicates if I have the destination position*/

bool loop = false ; /* this flag indicates that occured a loop */

bool IgnoredPacket = false ; /* indicates that the node doesnt have
                               the dest position*/

bool NoNeighborInArea = false; /* No neighbor in the flooding angle area */

bit trash = 0;
/* This procedure is to represent the internal behavior of the node . */
/* proctype InternalNode(){

bool nothing

// there can't be a void procedure in promela

} */

/* ===== NETWORK ===== */
/* This procedure represent the network communications among the nodes. */
proctype Network(){

/*
* Local variables chose randomly
*/
```

```
byte priorid;    /* Id of the prior node */
byte type;       /* Type of message */
/* byte trash = 0; */ /* necessary by if conditional structure */
```

```
atomic {
```

```
/* We will chose al parameters of the node in a random way.
```

```
- The ID
```

```
- The routing information age (destination timestamp)
```

```
- The flooding angle
```

```
    All this variables are chose in this way to guarantes that
all cases will be covered by the formal verification.
```

```
We chose as sample, to represent all possibles angles the angles
30, 45, 90, 135, 180.
```

We split the timestamp in tree ranges. The first is assinged with the angle 30 degrees, this represents new information. The second range is assigned to angles between 30 and 90 degrees, that represents old information, and the third range angles between 90 and 180 degrees, this represent the oldest information possible at the nodes.

We made a simplification, with relationtip the angle calculation. We will consider the age of the information directly proportional to the flooding angle. So we are fixing the r, of the formula, and providing directly the angle.

We will work with relative positions. If the previous node send the packet with the angle 30 degrees, this means that the previous node is between the actual node and the destination. The previous node is relatively before the actual node. If the degree is 90, the previous can stay at the same distance to the origin, at the same "level", no one is after the other. If the angle is greater than 90 the previous node can stay after the actual node. This means that the node could sent the message to back.

we can assume that node will be in the range because the DREAM always sends a message to its neighbours

```
*/
```

```
/*
 * receive the message
 */

medium?type(origin,destination,ttl,previousAngle)->

/* DEBUG */
printf("NET: Pkt_type:%d Origin:%d Destination:%d TTL:%d
Myid:%d PreviousAngle:%d \n",type,origin,destination,ttl,myid, previousAngle);

/*
 * Choosing my ID. We put the restriction in (myid =0) to avoid loop
 * if the angle is less than 90 degrees.As explained before.
 */
if
:: (previousAngle >= 90) -> myid = 0
:: (true) -> myid = 1
:: (true) -> myid = 2
:: else if
    :: (true) -> myid = 1
    :: (true) -> myid = 2
fi
fi->

if
:: (myid == 0) -> loop = true
:: else trash=1;
fi->

if
/* Im the destination */
:: (myid == destination ) ->
    if
        :: (sentAck == true) -> duplicatedMessage = true
        :: (sentAck == false) -> sentAck = true
    fi

/* Im the intermiate node */
:: else ->
```

```
if
  :: (HaveDestPosition == true) ->

  /*
  * Chose the information age
  */
  if
    :: (true) -> destinationTimestamp = 1
    :: (true) -> destinationTimestamp = 2
    :: (true) -> destinationTimestamp = 3
  fi->

  /*
  * Chose the angle
  */
  if
    :: (destinationTimestamp == 1) -> angle = 30
    :: (destinationTimestamp == 2) ->
      if
        :: (true) -> angle = 30
        :: (true) -> angle = 45
        :: (true) -> angle = 90
      fi
    :: (destinationTimestamp == 3) ->
      if /* one angle sample of each case */
        :: (true) -> angle = 30
        :: (true) -> angle = 90
        :: (true) -> angle = 135
        :: (true) -> angle = 180
      fi
    :: else trash = 2
  fi->

if /* if exist any neighbor in the angle area */

  /* not found*/
  :: (true) -> NoNeighborInArea = true ->
    IgnoredPacket = true

  /* found nodes at the area */
  :: (true) -> medium!type(origin,destination,(TTL-1),angle)
```

```
fi

    :: (HaveDestPosition == false)-> IgnoredPacket = true

fi /* havedestposition == true or false*/

fi/* Im the destination */

}/* atomic */

}/* Network */

/* ===== INIT ===== */

init{

    byte i; /* counter */

    byte dest; /* shift the row, because we put the
                Matrix of nodes in a unidimensional way*/

    byte Pkt_type; /* Type of the packet */

    byte Origin, Destination; /* Origin and Destination
                                read from channel */

    byte DestinationTimestamp, Angle; /* The timestamp of the destination
                                        information, flooding Angle */

    atomic {

/*
* We have fixed the Origin and the Detination to simplify
* the model and its analisis. This doesnt represent a lost of generality.
*/
        Origin      = 0;
        Destination = 2;

/*
* Chose the information age
*/
        if
            :: (true) -> DestinationTimestamp = 1
```

```
:: (true) -> DestinationTimestamp = 2
:: (true) -> DestinationTimestamp = 3
fi->

/*
 * Chose the angle
 */
if
  :: (DestinationTimestamp == 1) -> Angle = 30
  :: (DestinationTimestamp == 2) ->
    if
      :: (true) -> Angle = 30
      :: (true) -> Angle = 45
      :: (true) -> Angle = 90
    fi
  :: (DestinationTimestamp == 3) ->
    if /* one angle sample of each case */
  :: (true) -> Angle = 30
      :: (true) -> Angle = 90
      :: (true) -> Angle = 135
      :: (true) -> Angle = 180
    fi
  :: else trash = 2
fi->

/*
 * Choosing a random value to the Packet Type
 */

if
  :: (true) -> Pkt_type = 1 /* AckPacket */
  :: (true) -> Pkt_type = 2 /* Data Packet */
fi->

/* DEBUG */
printf("\tPkt_type:%d Origin:%d Destination:%d TTL:%d Angle:%d",
      Pkt_type, Origin, Destination, TTL, Angle) ->

medium!Pkt_type(Origin, Destination, TTL, Angle) ->

run Network()
```

```
 }/* atomic */
```

```
 }/* init */
```

A.7 LAR Verification Code

```

#define true 1
#define false 0
#define TTL 3
#define range 9 /* (nnodes * nnodes)*/
#define nnodes 3 /* Intermediates + Origin + Destination */

/*===== Assertions to test out =====*/

#define m0 (myid == 0)
#define m1 (myid == 1)
#define m2 (myid == 2)
#define IOT (InsideOutside == true)
#define IOF (InsideOutside == false)

#define Forwarded (Forward == true)
#define origin_isme (origin == myid)
#define destination_isme (destination == myid)
#define loop_teste (Loop == true)

/* Altered at 29/05/00 */

#define NotDellivery (ReachDestination == false)

/* Altered */
/* Reach More Than One Message in the Destination*/
#define receivedMTO (ReceivedMoreThanOne == true)

/* ===== */

bool InsideOutside; /* Inside Route Request Region */
bool InTheRange[range]; /* Inside the node range communication*/
/* we put the matrix[nnodes][nnodes] in
a unidimensional array*/
bool Loop = false; /* Reach a routing Loop */
bool Loop_int = false; /* Reach a routing Loop in intermediate node*/
bool RequestComplete = false; /* Complete the Request Process */
bool ReachDestination = false; /* Arrive at the destination */
bool ReceiveData = false; /* The data arrived at the destination*/

```



```

bool ReceivedMoreThanOne = false; /* This flag tell us if there was more than
                                     one message to the same node */

byte myid;                               /* choose in a randomicaly way the node id*/

byte  origin;        /* Origin of the message */
byte  ttl;           /* Time to Live of the message*/
byte  destination ; /* Destinatination of the message */

bool Forward = false; /*To indicate if the message has been forwarded*/

        /* packet types */
mtype = {RouteRequest, RouteReply, DataPacket};

        /* type,  origin, destination, TTL,  Prior node Id */
chan medium = [nnodes] of {mtype, byte,  byte,        byte, byte};

bool again;      /* This Flag is used to do the chooosing of sending once
                  or twice the message */

/* This procedure is to represent the internal behavior of the node .
   Now, we aren't using it effectively, because we think that it can
   cause a lot of overhead. But maybe in future */
/* proctype node(){

bool nothing
// there can't be a void procedure in promela

} */

/* This procedure represent the network communications among the nodes. */
proctype Network(){

/*
 * Local variables chose randomly
 */
byte priorid;    /* Id of the prior node */
byte type;       /* Type of message */
byte a;          /* Flag Variable to identify which assertion*/

atomic {

```

```
/*
 * Choosing a random value to "again" which, in a reality, chooses
 * if it sends 1 ou 2 copies of the message. When two copies
 * are sent it is done to emulate the broadcasting by choosing "again ==true"
 */

if
  :: (true) -> again = true
  :: (true) -> again = false
fi;

/*
 * Choosing a random value to my ID
 */
if
  :: (true) -> myid = 2
  :: (true) -> myid = 1
  :: (true) -> myid = 0
fi;

/*
 * receive the message
 */
medium?type(origin,destination,ttl,priorid);

printf("Pkt_type:%d Origin:%d Destination:%d TTL:%d PriorID:%d Myid:%d \n",
       type,origin,destination,ttl,priorid,myid);

/*
 * If the node is in the communication range
 */
if
  :: (InTheRange[(myid* nnodes + priorid)] == true) ->
  /*
   * Verification of the message type
   */
  if
    :: (type == RouteRequest) ->
```

```
if /* Im origin. Probably a loop */
  :: (origin == myid) ->

    if /* Ignore message. Ive sent a message to myself */
      :: (destination == myid)
        /* Don't continue to process the message. Loop. */
        :: else Loop = true
    fi

  /* Im dest. I will send a Route Reply */
  :: (destination == myid) ->

    if
      :: (ReachDestination == true) -> ReceivedMoreThanOne = true
      :: (ReachDestination == false) -> ReachDestination = true
    fi ->

  /* Send a Request Response */
  medium!RouteReply(destination,origin,TTL,myid) ->

  /*
  * Sending the message again or not
  */
  if
    :: (again == true) ->
      /* Send a Request Response */
      medium!RouteReply(destination,origin,TTL,myid)
    :: (again == false) -> again = false /* doing nothing */
  fi

  /*Im a intermediate node. I have to forward the packet */
  :: else

  if /* TTL haven't expired. So, we can forward the message */
    :: (ttl != 0) ->
      if
        :: (Forward == true) -> Loop_int = true
        :: (Forward == false) -> Forward = true ->
          medium!type(origin,destination,(ttl - 1),myid)->
        /*
        * Sending the message again or not
        */
      fi
    fi
  fi
```

```
        */
        if
            :: (again == true) ->
                medium!type(origin,destination,(ttl - 1),myid)
            :: (again == false) -> again = false /* doing nothing */
        fi ->

        run Network()

        fi
        /* TTL have expired. We need ignore the message */
        :: else -> a = 1 ;
    fi

fi /* message type is a route request */

:: (type == RouteReply) ->

if /* Im dest. The message was processed succesfully */
    /* The route request process was completed*/
    :: (destination == myid) -> RequestComplete = true

    /*Im origin of the message. Probably a loop*/
    :: (origin == myid) ->

    /*
    * Verify the TTL to know if a loop or not
    */
    if /* If the TTL was changed, it is a loop*/
        :: (ttl != TTL) -> Loop = true

        /*I read the packet that I ve just put on the pipe*/
        :: else -> a = 2
    fi /* If Im the origin */

    /*Im a intermiate node. Forward the packet*/
    :: else

    if /*if the message is not expired and the node is
        inside of the Route Request Region, we have to
        forward the message */
        :: ((ttl != 0) && (InsideOutside == true)) ->
```

```
        if
            :: (Forward == true) -> Loop_int = true
            :: (Forward == false) -> Forward = false ->
                medium!type(origin,destination,(ttl - 1),myid)->

            /*
            * Sending the message again or not
            */
            if
                :: (again == true) ->
                    medium!type(origin,destination,(ttl - 1),myid)
                :: (again == false) -> again = false /* doing nothing */
            fi ->

                run Network()

            fi
            /* I can't forward the packet*/
            :: else -> a = 3
        fi

    fi /* if the message is a Route Reply */

:: (type == DataPacket) ->
    if
        /*Im origin. Probably a loop*/
        :: (origin == myid) ->

            if /*Ignore message. Ive sent a message to myself
                or I heard a message that I sent*/
                :: (ttl == TTL)

                    /* A problem has occurred. I have to ignore the message*/
                    :: else Loop = true /*Ignore message*/
                fi /* if Im the origin */

                /*Im dest. I will receive and treat the packet*/
                :: (destination == myid) -> ReceiveData = true

                /*Im a intermediate node. Forward the packet */
                :: else ->

                    if /*if the message is not expired and the node is
```

```

        inside of the Route Request Region
        I have to forward the message*/
        :: ((ttl != 0) && (InsideOutside == true)) ->
            if
                :: (Forward == true) -> Loop_int = true
                :: (Forward == false)-> Forward = true->
                    medium!type(origin,destination,(ttl - 1),myid)->

                /*
                * Sending the message again or not
                */
                if
                    :: (again == true) ->
                        medium!type(origin,destination,(ttl - 1),myid)
                    :: (again == false) -> again = false /* doing nothing */
                fi ->

                run Network()

            fi

        /* A problem has occurred. I have to ignore the message*/
        :: else -> a = 4 /*Ignore message*/
    fi

    fi /* If the packet is a data packet */

    fi /* Verification of the message type*/

    /* I am not in the range. So Im not hearing the message */
    :: else -> a = 5
    fi /* the sender is in my range */

} /* atomic */
}/* Network */

init{

    byte i;          /* counter */
    byte desl ;     /* shift the row, because we put the
                    Matrix of nodes in a unidimensional way*/
    byte Pkt_type; /* Type of the packet */

```

```
int Origin, Destination; /* Origin and Destination
                           read from channel */

atomic{

  atomic{
    /*
     * We are force the node be in his own transmission range
     */
    i=0;
    desl=0;
    do
      :: (i < range) -> InTheRange[i + desl] = true ->
          i = i + nnodes ->
          desl = desl + 1
      :: (i >= range) -> break
    od;

    /*
     * we are setting up randomly the others values of the range
     */
    i = 1; /* it starts in 1 because 0 is a position
            of the first node to himself */
    do
      :: (InTheRange[i] == 0) ->
          if
            :: (true) -> InTheRange[i] = 0
            :: (true) -> InTheRange[i] = 1
          fi ->
          if
            :: (i == 8) -> break
            :: (i != 8) -> i = i + 1
          fi
      :: (InTheRange[i] == 1) ->
          if
            :: (i == 8) -> break
            :: (i != 8) -> i = i + 1
          fi
    od;
  } /* atomic */

  /*
```

```
* Choosing a random value to my Origin
*/
if
  :: (true) -> Origin = 0
  :: (true) -> Origin = 1
  :: (true) -> Origin = 2
fi;

/*
* Choosing a random value to my Destination
*/
if
  :: (true) -> Destination = 0;
  :: (true) -> Destination = 1;
  :: (true) -> Destination = 2;
fi;

/*
* Choosing a random value to the Packet Type
*/

if
  :: (true) -> Pkt_type = 1 /* Route Request */
  :: (true) -> Pkt_type = 2 /* Route Reply */
  :: (true) -> Pkt_type = 3 /* Data Packet */
fi;

/* DEBUG */
printf("\tPkt_type:%d Origin:%d Destination:%d TTL:%d PriorID:%d \n",
       Pkt_type, Origin, Destination, TTL, Origin);

/*
* Start the processing. First send a message to anyone
* and continue after that. We are using only one message
* and only one intermediate node because we think that with
* only one i-node we can represent a lot of i-nodes.
*/
atomic {medium!Pkt_type(Origin, Destination, TTL, Origin) ->

  /*
  * Sending the message again or not
  */
```



```
    if
      :: (again == true) ->
          medium!Pkt_type(Origin, Destination, TTL, Origin)
      :: (again == false) -> again = false /* doing nothing */
    fi ->

    run Network()

} /* atomic */

}/* init */
```

802.11	The IEEE standard for wireless connectivity at 1 Mbps and 2 Mbps in the 2.4 GHz band
Ad hoc	A group of wireless devices communicating directly with each other (peer-to-peer) without the use of an access point
Assertion	A directive to a tool telling it what to do with a property. Assertions are properties that are evaluated within an execution engine: a simulator, emulator, or formal analysis tool. An assertion provides a monitor that ensures a property holds once, always holds, or never holds during verification of the design
Backbone	The part of a network that connects most of the systems and networks together, and handles the most data
Backbone	A cable to which multiple nodes or workstations are attached
Bandwidth	The amount of data you can send through a channel (measured in bits per second)
Broadband	Transmission by modulated carrier - Data over TV cable is broadband, DSL is not. In cable transmission a high frequency carrier signal is modulated by data. In DSL data signals are sent as changes in voltage on the wire. DSL should be called "high data rate", not broadband
Cell	The geographic region that is serviced by one base station (either analog cellular or digital)
Counter-Example	Stimulus sequence from a legal design state that violates an assertion
CSMA/CA	Carrier Sense Multiple Access Collision Avoidance is a network access method in which each device signals its intent to transmit before it actually does so. This prevents other devices from sending information, thus preventing collisions from occurring between signals from two or more devices. This is the access method used by LocalTalk
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access Collision Detection is a network access method in which devices that are ready to transmit data first check the channel for a carrier. If no carrier is sensed, a device can transmit. If two devices transmit at once, a collision occurs and each computer backs off and waits a random amount of time before attempting to retransmit. This is the access method used by Ethernet

Download	To receive a file transmitted over a network
Flooding	a simple routing algorithm in which every incoming packet is sent through every outgoing link
Formal Verification (FV)	The use of mathematical models and analysis to functionally verify design behavior. Also, a product (0-In Formal Verification) that uses formal techniques for model checking
Global Positioning System (GPS)	A worldwide, satellite-based radio navigation system providing three-dimensional position, velocity and time information to users having GPS receivers anywhere on or near the surface of the Earth
Hardware	The physical aspect of computers, telecommunications, and other information technology devices
Hierarchical networks	A network in which a host controls network communications and processing
Hop	Transmission, or link, between two neighbor nodes
Host	Computer that controls network communication in a hierarchical network
IEEE	Institute of Electrical and Electronic Engineers
Internet	A global network that incorporates networks belonging to the United States government, academic institutions, and other organizations
Internet Protocol (IP)	A protocol used to send data over a network
Media Access Control (MAC)	The unique address that a manufacturer assigns to each networking device
MANET	mobile ad hoc network, sometimes called a mobile mesh network, is a self-configuring network of mobile devices connected by wireless links
Mesh Network	architecture in which each node has a dedicated connection to all other nodes. Node A network-access point. Examples include terminals and computers Real time A transmission or transaction that occurs immediately or in an extremely short period of time. A telephone conversation occurs in real time; correspondence through mail does not
Middleware	An intermediate software component located on the wired network between the wireless appliance and the application or data residing on the wired network

Mobile IP	A protocol developed by the Internet Engineering Task Force to enable users to roam to parts of the network associated with a different IP address
Mobility	Ability to continually move from one location to another
Network	A series of computers or devices connected for the purpose of data sharing, storage, and/or transmission between users
Node	A network junction or connection point, typically a computer or work station
Proof	The result of static formal verification when it can be determined that an assertion is never violated from the given initial state
Property	A concise statement about a specific intended behavior of a design. Properties provide concise, mathematically precise descriptions of behavior about the design, or that constrain the operating environment of the block. In addition, properties can also describe coverage points or scenarios that must be exercised by the verification process. Properties can specify functionality, timing, or any other aspect of the design. The term assertion is often used interchangeably with property
Roaming	The ability to take a wireless device from one access point's range to another without losing the connection
Roaming	Traveling from the range of one access point to another
Router	A networking device that connects multiple networks together, such as a local network and the Internet. Routing (or routening) is the process of selecting paths in a network along which to send network traffic
Software	Instructions for the computer. A series of instructions that performs a particular task is called a "program"
TCP/IP	Transport Control Protocol/Internet Protocol. Refers to the Internet Protocols, a set of protocol originally developed for the United States government. Because the Internet Protocols have been implemented on a wide variety of computers, they are often used in networks that interconnect disparate systems
Throughput	The amount of data moved successfully from one node to another in a given time period
Time-to-live	Valid time of a message, when this value reaches the limit the message is removed from the network. Normally this value is specified in number of hops

Upload	To transmit a file over a network
Wi-Fi Wireless Fidelity	Wi-Fi is meant to be used generically when referring of any type of 802.11 network, whether 802.11b, 802.11a, dual-band, etc. The term is promulgated by the Wi-Fi Alliance

This appendix presents a small guide of the syntax of PROMELA. The syntax examples and explanations are retrieved from the Basic Spin Manual from the webpage <http://spinroot.com/spin/Man/Manual.html>, last visited in December 2009.

A Promela model consist of:

- type declarations
- channel declarations
- global variable declarations
- process declarations
- init process

A process is defined by a proctype definition and executes concurrently with all other processes, independently of speed or behavior communicating with other processes using either global (shared) variables or channels. There may be several processes of the same type each one with its own local state:

- process counter (location within the proctype)
- contents of the local variables

Variables are used to store either global information about the system as a whole, or information local to one specific process, depending on where the declaration for the variable is placed. The declarations:

- `bool flag;`
- `int state;`
- `byte msg;`

define variables that can store integer values in three different ranges. The scope of a variable is global if it is declared outside all process declarations, and local if it is declared within a process declaration.

8.2 Data Types

The table below summarizes the basic data types, sizes, and typical value ranges on a 32-bit wordsize computer.

The names `bit` and `bool` are synonyms for a single bit of information. An defined type, `mtype`, variable can be assigned symbolic values that are declared in an `mtype = ...` statement, to be discussed below.

Typename	C-equivalent	Typical Range
bit or bool	bit-field	0..1
byte	uchar	0..255
short	short	$-2^{15} - 1..2^{15} - 1$
int	int	$-2^{31} - 1..2^{31} - 1$

Table 8.1: Data Types

8.3 Array Variables

Variables can be declared as arrays. For instance,

```
byte state[N]
```

declares an array of N bytes that can be accessed in statements such as

```
state[0] = state[3] + 5 * state[3*2/n]
```

where n is a constant or a variable declared elsewhere.

8.4 Process Types

The state of a variable or of a message channel can only be changed or inspected by processes. The behavior of a process is defined in a proctype declaration. The following, for instance, declares a process with one local variable state.

```
proctype A()
{ byte state;

state = 3
}
```

The process type is named A. The body of the declaration is enclosed in curly braces. The declaration body consists of a list of zero or more declarations of local variables and/or statements. The declaration above contains one local variable declaration and a single statement: an assignment of the value 3 to variable state. The semicolon is a statement separator (not a statement terminator, hence there is no semicolon after the last statement). Promela accepts two different statement separators: an arrow ‘->’ and the semicolon ‘;’. The two statement separators are equivalent. The arrow is sometimes used as an informal way to indicate a causal relation between two statements. Consider the following example.

```
byte state = 2;

proctype A()
{ (state == 1) -> state = 3
}
```

```
proctype B()
{ state = state - 1
}
```

In this example we declared two types of processes, A and B. Variable state is now a global, initialized to the value two.

8.5 Atomic Sequences

In Promela there is also another way to avoid the test and set problem: atomic sequences. By prefixing a sequence of statements enclosed in curly braces with the keyword `atomic` the user can indicate that the sequence is to be executed as one indivisible unit, non-interleaved with any other processes. It causes a run-time error if any statement, other than the first statement, blocks in an atomic sequence. This is how we can use atomic sequences to protect the concurrent access to the global variable state in the earlier example.

```
byte state = 1;

proctype A()
{ atomic {
  (state==1) -> state = state+1
}
}

proctype B()
{ atomic {
  (state==1) -> state = state-1
}
}

init
{ run A(); run B()
}
```

In this case the final value of state is either zero or two, depending on which process executes. The other process will be blocked forever.

8.6 Message Passing

Message channels are used to model the transfer of data from one process to another. They are declared either locally or globally, for instance as follows:

```
chan qname = [16] of { short }
```


This declares a channel that can store up to 16 messages of type `short`. Channel names can be passed from one process to another via channels or as parameters in process instantiations. If the messages to be passed by the channel have more than one field, the declaration may look as follows:

```
chan qname = [16] of { byte, int, chan, byte }
```

8.7 The statement

```
qname!expr
```

sends the value of expression `expr` to the channel that we just created, that is: it appends the value to the tail of the channel.

```
qname?msg
```

receives the message, it retrieves it from the head of the channel, and stores it in a variable `msg`.

8.8 Control Flow

Between the lines, we have already introduced three ways of defining control flow: concatenation of statements within a process, parallel execution of processes, and atomic sequences. There are three other control flow constructs in Promela to be discussed. They are case selection, repetition, and unconditional jumps.

8.8.1 Case Selection

The simplest construct is the selection structure. Using the relative values of two variables `a` and `b` to choose between two options, for instance, we can write:

```
if
:: (a != b) -> option1
:: (a == b) -> option2
fi
```

The selection structure contains two execution sequences, each preceded by a double colon. Only one sequence from the list will be executed. A sequence can be selected only if its first statement is executable. The first statement is therefore called a guard.

8.8.2 Repetition

A logical extension of the selection structure is the repetition structure. We can modify the above program as follows, to obtain a cyclic program that randomly changes the value of the variable up or down.

```
byte count;
```

```
proctype counter()
{
do
:: count = count + 1
:: count = count - 1
:: (count == 0) -> break
od
}
```

Only one option can be selected for execution at a time. After the option completes, the execution of the structure is repeated. The normal way to terminate the repetition structure is with a break statement.

Bibliography

- [1] Oskar Wibling, Joachim Parrow, and Arnold Pears, Ad Hoc Routing Protocol Verification Through Broadcast Abstraction, Proceedings of the 25th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), Taipei, Taiwan, October 2005.
- [2] K. Bhargavan, D. Obradovic, C. A. Gunter, Formal verification of standards for distance vector routing protocols, Journal of the ACM, 538-576, Volume 49, Number 4, y 2002.
- [3] Sibusisiwe Chiyangwa and Marta Kwiatkowska. A timing analysis of AODV, 7th IFIP FMOODS, June 2005.
- [4] D. Câmara, A. A. F. Loureiro, F. Filali, Methodology for Formal Verification of Routing Protocols for Ad Hoc Wireless Networks, IEEE GLOBECOM 2007, Washington, DC, November, 2007.
- [5] D, Câmara, C. F. Santos, A. A . F. Loureiro, Formal Verification of Routing Protocols for Ad hoc Networks, Brazilian Symposium on Computer Networks, SC, Brazil, 2001. (In Portuguese)
- [6] S. Basagni, I. Chlamtac, V. Syrotiuk, and B. Woodward, A Distance Routing Effect Algorithm For Mobility, MobiCom'98, Dallas, TX, 1998.
- [7] Y. Ko and N. H. Vaidya, Location-Aided Routing (LAR) Mobile Ad Hoc Networks, MobiCom'98, Dallas, TX, 1998.
- [8] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum, L. Viennot, Optimized Link State Routing Protocol for Ad Hoc Networks, IEEE INMIC Pakistan 2001.
- [9] E. M. Clarke, O. Grumberg, and D. A. Peled. Model Checking. MIT Press, 1999.
- [10] C. Kern and M. R. Greenstreet. Formal verification in hardware design: a survey. ACM Transactions on Design Automation of Electronic Systems, 4(2):123-193, April 1999.
- [11] A. Hall, Seven Myths of Formal Methods, IEEE Software, Sept 1990.
- [12] O. Wibling, Ad Hoc Routing Protocol Validation, Licentiate Thesis 2005-004, Dept of Info Technology, Uppsala University, Sweden, 2005.

-
- [13] C. Yuan, J. Billington, An Abstract Model of Routing in Mobile Ad Hoc Networks, Sixth Workshop and Tutorial on Practical Use of CPN and the CPN Tools, Aarhus, Denmark, 2005.
 - [14] D. Obradovic, Formal Analysis of Convergence of Routing Protocols, Ph.D. Thesis Proposal, Department of Computer and Information Science, University of Pennsylvania, Nov. 2000.
 - [15] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward, A distance routing effect algorithm for mobility (DREAM), in ACM/IEEE Mobicom 98, pages 76 - 84.
 - [16] T. Clausen, P. Jacquet, Optimized Link State Routing Protocol (OLSR), Request for Comments: 3626, October 2003.
 - [17] G. J. Holzmann, The model checker SPIN. IEEE Trans. on Software Eng., 23(5), May 1997.
 - [18] G.J. Holzmann Design and Validation of Computer Protocols. Englewood Cliffs, N.J.: Prentice Hall, 1991.
 - [19] F. J. Lin, P. M. Chu, M. T. Liu, Protocol verification using reachability analysis: the state space explosion problem and relief strategies, SIGCOMM '87, ACM Press, 1988.
 - [20] Jonathan P. Bowen and Michael G. Hinchey Seven More Myths of Formal Methods, IEEE Software, July 1995.
 - [21] Department Of Defense Standard: Department Of Defense Trusted Computer System Evaluation Criteria (Aka. The Orange Book). DoD 5200.28-STD; Supersedes; CSC-STD-001-83, dtd 15 Aug 83; Library No. S225,711.
 - [22] R. de Renesse, A. H. Aghvami, Formal Verification of Ad-Hoc Routing Protocols Using SPIN Model Checker, 12th Mediterranean Electrotechnical Conference, Croatia, 2004.
 - [23] C. Xiong, T. Murata, and J. Tsai, Modeling and Simulation of Routing Protocol for Mobile Ad Hoc networks Using Colored Petri Nets, Research and Practice in Information Technology, Vol. 12, pp.145-153, Australian Computer Society, 2002.
 - [24] Kristensen, Lars Michael, Jensen, Kurt, Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad Hoc Networks, Integration of Software Specification Techniques for Applications in Engineering, V. 3147 of Lecture Notes in Computer Science, Springer-Verlag, September 2004.
 - [25] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, Model Checking, MIT Press, 1999.
 - [26] S. Hendriex and L. Claesen, A symbolic core approach to the formal verification of integrated mixed-mode applications, EDTC '97 European conference on Design and Test, 1997.
 - [27] Patrice Godefroid, An Approach to the State-Explosion Problem, Ph.D. thesis, University of Liege, Computer Science Department, 1994.

- [28] G.J. Holzmann and D. Peled. An improvement in formal verification. In Proc. 7th IFIP WG 6.1 International Conference on Formal Description Techniques, October 1994.
- [29] Sergey Berezin, Sérgio Campos, and Edmund M. Clarke. Compositional reasoning in model checking. In *Compositionality: The Significant Difference: International Symposium*, V. 1536 of Lecture Notes in Computer Science, Springer Verlag, September 1997.
- [30] E. M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction, *ACM Transactions on Programming Languages and Systems*, 16(5):1512-1542, September 1994.
- [31] E. Allen Emerson and Richard J. Trefler, *From Asymmetry to Full Symmetry: New Techniques for Symmetry Reduction in Model Checking*, Conference on Correct Hardware Design and Verification Methods, p. 142-156, 1999.
- [32] E. M. Clarke, E.A. Emerson, S. Jha, and A.S. Sistla. Symmetry reductions in model checking. V. 1427 of Lecture Notes in Computer Science, Springer Verlag, June/July 1998.
- [33] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum, L. Viennot, *Optimized Link State Routing Protocol for Ad Hoc Networks*, IEEE INMIC Pakistan 2001.
- [34] T. Clausen, P. Jacquet, *Optimized Link State Routing Protocol (OLSR)*, Request for Comments: 3626, October 2003.
- [35] Edmund Clarke, *Model Checking: My 25 year quest to overcome the state-explosion problem*, 25 Years of Model Checking Symposium, The 2006 Federated Logic Conference, Seattle, Washington, August 10 - 22, 2006.
- [36] M. Aagaard, M. E. Leeser, and P. J. Windley. *Toward a super duper hardware tactic*, 6th International Workshop, HUG'93, Vancouver, B.C., August 11-13 1993.
- [37] T. Murata, *Petri Nets: Properties, Analysis and Applications* Proceedings of the IEEE, pp. 541-580, Vol. 77, No 4, April, 1989.
- [38] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*, Monographs in Theoretical Computer Science, Springer-Verlag, 1997.
- [39] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson and Wang Yi, *Uppaal - a Tool Suite for Automatic Verification of Real-Time Systems*, In Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Jersey, Oct. 1995.
- [40] Madanlal Musuvathi, David Y. W. Park, Andy Chou, Dawson R. Engler, David L. Dill: *CMC: A Pragmatic Approach to Model Checking Real Code*, 5th Symposium on Operating System Design and Implementation, USENIX Association, Massachusetts, Dec. 2002.
- [41] Irfan Zakiuddin, Michael Goldsmith, Paul Whittaker, Paul H. B. Gardiner: *A Methodology for Model-Checking Ad-hoc Networks*, Lecture Notes in Computer Science, Volume 2648, Springer Verlag, May 2003.

-
- [42] G. Ács, L. Buttyán, and I. Vajda, Provable Security of On-Demand Distance Vector Routing in Wireless Ad Hoc Networks, Second European Workshop on Security and Privacy in Ad Hoc and Sensor Networks (ESAS 2005) Visegrád, Hungary, July 13-14, 2005.
- [43] Satyaki Das and David L. Dill. Counter-Example Based, Predicate Discovery in Predicate Abstraction. Formal Methods in Computer-Aided Design, Portland, Oregon, November, 2002.
- [44] Richard Ogier. Topology dissemination based on reverse-path forwarding (TBRPF): Correctness and simulation evaluation, Technical report, SRI International, October 2003.
- [45] S. Das, A. Nandan, G. Pau M.Y. Sanadidi and M. Gerla, SPAWN: Swarming Protocols for Vehicular Ad Hoc Wireless Networks, Proceedings of the First ACM International Workshop on Vehicular Ad Hoc Networks (VANET 2004), MOBICOM 2004, Berkeley, 2004.
- [46] A. Nandan, S. Das, G. Pau M.Y. Sanadidi and M. Gerla —Cooperative Downloading in Vehicular Ad Hoc Wireless Networks, Proceedings of IEEE/IFIP International Conference on Wireless On demand Network Systems and Services , St. Moritz, Switzerland, Jan 2005.
- [47] R. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling a threetier architecture for sparse sensor networks. Sensor Network Protocols and Applications. IEEE, 2003.
- [48] Goodman, D.J. Borrás, J. Mandayam, N.B. Yates, R.D, INFOSTATIONS: a new system model for data and messaging services, IEEE 47th Vehicular Technology Conference, Phoenix, AZ, USA, 1997.
- [49] Zong Da Chen, H.T.Kung, and Dario Vah. Ad hoc relay wireless networks over moving vehicles on highways. In MobiHoC, 2001.
- [50] Qun Li and Daniela Rus. Sending messages to mobile users in disconnected ad-hoc wireless networks. In Proceedings of the sixth annual international conference on Mobile computing and networking, pages 44–55. ACM Press, 2000.
- [51] Briesemeister, L. and Hommel, G., Role-Based Multicast in Highly Mobile but Sparsely Connected Ad Hoc Networks, in Proceedings of the First Annual Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC), Boston, MA, USA, August 2000.
- [52] Gavrilovich, C. D., Broadband Communication on the Highways of Tomorrow, in IEEE Communications Magazine, April 2001.
- [53] J. Härri, M. Fiore, F. Fethi, and C. Bonnet, VanetMobiSim: generating realistic mobility patterns for VANETs, in Proc. of the 3rd ACM International Workshop on Vehicular Ad Hoc Networks (VANET'06), September 29, 2006, Los Angeles, USA.
- [54] Swarup Acharya, Michael J. Franklin, and Stanley B. Zdonik. Balancing push and pull for data broadcast. In ACM SIGMOD, pages 183-194, 1997.
- [55] DimitriosKatsaros and YannisManolopoulos. Web caching in broadcastmobile wireless environments. IEEE Internet Computing, 08(3):37-45, 2004.

-
- [56] Chi-Jiun Su and Leandros Tassiulas. Joint broadcast scheduling and user's cache management for efficient information delivery. *Wirel. Netw.*, 6(4):279-288, 2000.
- [57] Qiu Fang, Susan V. Vrbsky, Yu Dang, and Weigang Ni. A pull-based broadcast algorithm that considers timing constraints. In *ICPPW 04*, pages 46-53, IEEE Computer Society, Washington, DC, USA, 2004.
- [58] Liviu Iftode Tamer Nadeem, Pravin Shankar. A comparative study of data dissemination models for vanets. In *Proceedings of the 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MOBIQUITOUS 2006)*, July 2006.
- [59] Nikolaos Frangiadakis and Nick Roussopoulos. Caching in mobile environments: A new analysis and the mobile - cache system. In *IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, Athens, Greece, September 2007.
- [60] M. Tariq, M. Ammar, E. Zegura. Message Ferry Route Design for Sparse Ad hoc Networks with MobileNodes. *ACM Mobihoc 2006*. May 2006 22-27, Florence. Italy.
- [61] P. Juang and H. Oki and Y. Wang and M. Martonosi and L. Peh and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *ASPLOS*, San Jose, CA, Oct 2002.
- [62] Nikolaos Frangiadakis, Daniel Câmara, Fethi Filali, Antonio Alfredo F. Loureiro, Nick Roussopoulos, Virtual access points for vehicular networks *Mobilware 2008*, 1st International Conference on MOBILE Wireless MiddleWARE, Operating Systems, and Applications, Innsbruck, Austria, February 12th-15th, 2008.
- [63] Charles E. Perkins and Elizabeth M. Royer. Adhoc ondemand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications*, pages 90-100, Los Alamitos, California, February 1999.
- [64] G. J. Holzmann, An analysis of bitstate hashing, *Formal Methods in Systems Design*, November 1998.
- [65] P. Khengar, Wireless adaptive routing protocol, PhD Dissertation, King's College London, Centre for Telecommunication Research, Dec. 2003
- [66] Mingliang Jiang, Jinyang Li and Y.C. Tay. Cluster Based Routing Protocol (CBRP). Internet Draft draft-ietf-manet-cbrp-spec-01.txt, August 1999.
- [67] Manel Guerrero Zapata and N. Asokan Securing Ad-Hoc Routing Protocols In *Proceedings of the 2002 ACM Workshop on Wireless Security (WiSe 2002)*, pages 1-10. September 2002.
- [68] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E. Belding-Royer. A secure routing protocol for ad hoc networks. In *Proceedings of the International Conference on Network Protocols (ICNP)*, 2002.

- [69] O. Wibling, J. Parrow, and A. Pears, Automatized Verification of Ad Hoc Routing Protocols, Proc. 24th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004), Madrid, Spain, 27-30 Sept. 2004.
- [70] Christian Tschudin, Richard Gold, Olof Rensfelt, and Oskar Wibling. LUNAR - A Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation. In Proceedings of Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN), St. Petersburg, Russia, February 2004.
- [71] Sibusisiwe Chiyangwa and Marta kwiatkowska, Analysing Timed Properties of AODV with UPPAAL, University of Birmingham, School of Computer Science, Technical Report CSR-04-4, March 2004.
- [72] R. Ogier, F. Templin, and M. Lewis, Topology Dissemination Based on Reverse-Path Forwarding, draft-ietf-manet-tbrpf-11.txt, Internet-Draft, October 2003.
- [73] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6), December 1998. RFC 2461.
- [74] Satyaki Das and David L. Dill. Successive approximation of In Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science, pages 51-60. IEEE Computer Society, 2001. June 2001, Boston, USA.
- [75] Shuvendu K. Lahiri Randal E. Bryant, Predicate Abstraction with Indexed Predicates, eprint arXiv:cs/0407006, ARXIV, publication date 07/2004.
- [76] Virginia Gold, ACM Turing Award Honors Founders of Automatic Verification Technology, ACM, <http://www.acm.org/press-room/news-releases/turing-award-07/>, collected Feb. 2008.
- [77] R. P. Kurshan, Verification Technology Transfer, 25 Years of Model Checking - History, Achievements, Perspectives. Lecture Notes in Computer Science 5000 Springer 2008
- [78] Orna Grumberg and David Long, Model checking and modular verification, ACM Transactions on Programming Languages and Systems, 16(3):843-871, May 1994
- [79] A. Pnueli, In transition for global to modular temporal reasoning about programs, In K. R. Apt, editor, Logics and Models of Concurrent Systems, volume 13 of NATO ASI series. Series F, Computer and system sciences, Springer-Verlag, 1984
- [80] E.M. Clarke, S. Berezin, and S. Campos, Compositional Reasoning in Model Checking, Lecture Notes in Computer Science 1536, pp. 81-103, 1998
- [81] N. Sinha, Automated Compositional Analysis for Checking Component Substitutability, Doctoral Thesis, UMI Order Number: AAI3289953, Carnegie Mellon University, 2007
- [82] Gerard J. Holzmann, Basic Spin Manual, AT&T Bell Laboratories, Murray Hill, New Jersey

-
- [83] G. J. Holzmann. An analysis of bitstate hashing. In Proc. 15th International Conference on Protocol Specification, Testing, and Verification, INWG/IFIP, pages 301–314, Warsaw, Poland, 1995
- [84] D. Kroening and S. A. Seshia, Formal verification at higher levels of abstraction, In Proceedings of the 2007 IEEE/ACM international Conference on Computer-Aided Design, San Jose, California, Nov., 2007
- [85] Yohan Boichut, Pierre-Cyrille Héam, Olga Kouchnarenko, Automatic Verification of Security Protocols Using Approximations, Rapport De Recherche Inria, RR-5727, inria-00070291, Oct. 2005
- [86] Paolo Baldan, Andrea Corradini, and Barbara König. Unfolding-based verification for graph transformation systems. In Proc. of UniGra 03: Uniform Approaches to Graphical Specification Techniques, Warsaw, 2003
- [87] P. T. Breuer and S. Pickin, Verification in the Large via Symbolic Approximation, In Proceedings of the Second international Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISOLA, IEEE Computer Society, Washington, DC, November, 2006