Formal Verification of Routing Protocols for Wireless Ad Hoc Networks

Daniel Câmara
Department of Computer Science, Federal University of Minas Gerais,
Caixa Postal 702, 30123-970, Belo Horizonte, Minas Gerais, Brazil,
e-mail: danielc@dcc.ufmg.br

Antonio A. F. Loureiro
Department of Computer Science, Federal University of Minas Gerais,
Caixa Postal 702, 30123-970, Belo Horizonte, Minas Gerais, Brazil,
e-mail: loureiro@dcc.ufmg.br

Fethi Filali
Institut Eurécom,
2229 Route des Crêtes, BP 193 F-06560 Sophia-Antipolis France,
e-mail: fethi.filali@eurecom.fr

Abstract 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 wireless network. However, a difficult problem is how to guarantee these desirable qualities. Neither simulations nor testbed implementations can ensure the quality required for these protocols. 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 chapter presents the main tools, proposals and techniques available to perform formal verification of routing algorithms for wireless ad hoc networks.

1 Introduction

This chapter discusses the importance of applying formal verification in the development of routing algorithms for wireless ad hoc networks. It also presents a concise description of some of the most important proposals on this field.

We start by answering the following two questions: "What is a formal verification technique and 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 manly formal verification, it 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 quite easy and 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 chapter 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.

Next section discusses the main formal verification techniques and their main variants and problems. After that the section *Tools* presents some of the main available tools to perform formal verification of computer systems. The section *Thoughts for Practitioners* presents and exemplifies a simple and interesting way to perform formal verification. After that the section *Routing Verification Proposals* presents a concise view of some of the more important works on the field. After that the sections *Directions for Future Research* and *Conclusions* present some thoughts about future research on the field and final comments about the chapter.

2 Background

Formal verification is the process of verifying, through 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 notice that formal verification is not a substitute for testing or simulation. These three quality assurance techniques are much more 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 some parameter can be varied and the 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 sometimes better [20].

Notice also that neither formal verification nor testing can guarantee that the system is perfect [11]. As Edsger W. Dijkstra said once about testing, "Program testing can best show the presence of errors but never their absence". In the same way, formal verification can 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

There are basically three kinds of automated formal verification techniques, namely, model checking, theorem proving and equivalence checking. Model checking is a method to verify if a formally modeled system satisfies a given property [9]. Theorem proving technique uses mathematical methods, such as axioms and 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 will discuss these techniques, but it is worth to remember that even though they propose automated solutions, neither approach works without some degree of human assistance. For example, theorem proving sometimes requires advice of which properties worth to verify. Model checkers, on the other hand, can quickly get stuck when checking millions of useless states and human guidance can be handy.

- **Model Checking**: Model checking verifies, using an algorithm, if a given model is in accordance with the specification. The model is normally pro-

grammed 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. Fig. 1 presents the model checking approach. The tool receives as input the system model and the desired/undesired property to be checked. The output is the answer whether the system holds or not the requested property. In the last case, it is common to provide 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 accomplish this task, a model checking tool uses a combinatorial amount of states to represent the system. In another words, the number of states required to represent a system increases exponentially with its size, leading to a problem known as state explosion, discussed later in the Section *State Explosion Problem and Remedies*.
The success of the model checking verification depends on the user's expertise. Building a good model is a tradeoff between representing the important points of the system and decreasing the size of the model to avoid the state space problem. In this process, the model designer must be really careful to not remove fundamental system characteristics and, at the same time, reduce the system complexity so the model is feasible to be verified.

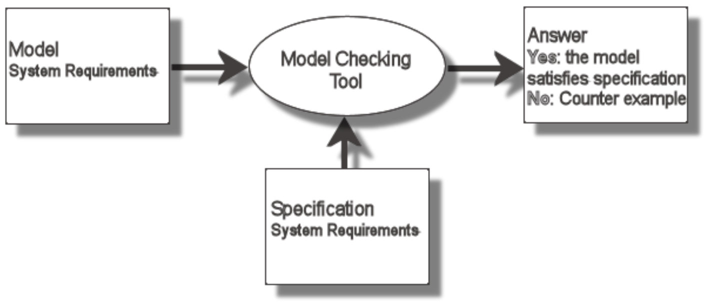

Fig. 1 - The Model checking approach .

- **Theorem Proving**: Theorem proving involves verifying the truth of mathematical theorems postulated about the design. Theorem proving is like any 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. Often the theorem proving tool 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 people able to guide the tool through the right path.

- **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 that two representations of a circuit present exactly the same behavior. In this case, typically, the gate-level implementation is compared with its representation at a higher level, Register Transfer Level (RTL). However, in general, equivalence checking can work well for two structurally similar designs.
  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 so it is difficult to determine the exact cause of the difference.

2.2 The state explosion problem and remedies

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. [26]:

- **Symbolic representation**: This technique refers to the use of compact data structures to represent the state space [27], 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 [40]. In this case it would not even be possible to store explicitly all time points, regardless the available amount of memory [12].
- **Partial order reduction**: It is based on the principle [28] 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 isolated from each other. Partial order reduction is about to analyze the processes execution and exploit the com-

mutativity 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 [29].

- **Compositional reasoning**: In short, it is the decomposition of the system into components which are verified apart from the other components [30]. Composing over 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 [31]. 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, 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 [32]. Symmetry reduction [33] 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.

3 Tools

The automatic verification of protocols is intrinsically linked to the software tools. This section briefly presents some of the most used formal verification tools available. All these are general tools and can be applied to a number of different applications to verify a broad range of systems.

- **HOL - Higher Order Logic**: HOL [37] 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.

- **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.
  Instead of doing the verification directly on the PROMELA code, to improve its performance, SPIN generates C code from the model. 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.
- **CPN – Coloured Petri Nets**: Petri Nets [38] 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) [39]. Unlike standard Petri nets, where tokens are indistinguishable, in a Coloured Petri Net, every token has a value. This make 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 [24] and ERDP [25].
- **UPPAAL**: UPPAAL [40] 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 failures in the modeled system. However its full power is shown when automati-

cally 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.

4 Thoughts for Practitioners

The use of formal verification techniques does not need to be difficult or extremely complex. Even simple approaches can lead to good and useful results. This section presents and exemplifies a simple, yet powerful, way to formally verify routing protocols for wireless networks. The technique was initially presented in [5] and improved in [4]. In a nutshell, the technique is basically a list of procedures to be followed when verifying protocol. The method encourages the utilization of standard formal verification techniques, some of them presented in the section *State Explosion Problem and Remedies*, and serves as a guide for newbies. Algorithm 1 presents the basic steps of the methodology described in [4].

The method is quite general and its steps may be useful to verify different protocols for wireless networks. In contrast to the work presented in [3], this specific method focuses on qualitative aspects of the protocols rather than on quantitative ones [4]. It is indicated to those who are interested in identify specification problems such as routing loops, packet delivery failures, unexpected reception of messages and other “pathological” cases not treated in the protocol specifications. For those interested in quantitative aspects, such as max/min number of messages, the particular behavior of a given topology or timing aspects, other methods such as [2, 3], discussed later in this chapter, are more appropriated.

As general advices when implementing this technique, we can highlight two points: first, build a simpler model and improve over it. Second, make sure to model all possible relations of the system under consideration. Starting with a simple verifiable code makes the work easier and with faster results. This also helps in filtering false positive results. Whenever a property is verified, the presented result must be analyzed in order to determine whether it is a failure in the algorithm or in the model. Unfortunately, up to now, this task can not be done automatically and following traces of complex models can be very hard.

Regarding the second advice, when using a model checker tool, one of the main concerns is to be careful enough to model all relations that can happen in the real world. Model checkers normally work performing a search through all possible states trying to find a condition that satisfies, or denies the requested property. For these tools it does not really matter if the verified property occurs in 99% of the cases or just in one single case. When the property does not hold, the tool finds this case and, often, presents the scenario where it occurs. After that, it is up to the system designer to analyze the scenario and figure out if the presented counterexample is a real problem or if it is a failure in the model.

```
1:   Acquire the information needed 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 message)
7:       For (each kind of node) {
8:           Specify the semantics of the message to the node;
9:       } // For
10:  }// 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)
15:      Creates an algorithm or an state machine representation to understand it better;
16:  } // 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)
20:      Analyses and verify the model
21:      For (each error found in the model)
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:      } // For
27:      Increase the model complexity;
28:  } // While
29:  Identify and isolate verified procedures to be used in other protocols.
```

Algorithm 1 – High level algorithm of the proposed method [4].

4.1 Case Study

This section presents the methodology applied to OLSR protocol [34], 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 [35], in order to meet our didactical objectives, given its simplicity, we use the original version [34].

4.1.1 Understanding the protocol

The Optimized Link State Routing Protocol (OLSR) [34] 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, rebroadcast them to other MPR (multi-point relay) nodes. The broadcasted 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.

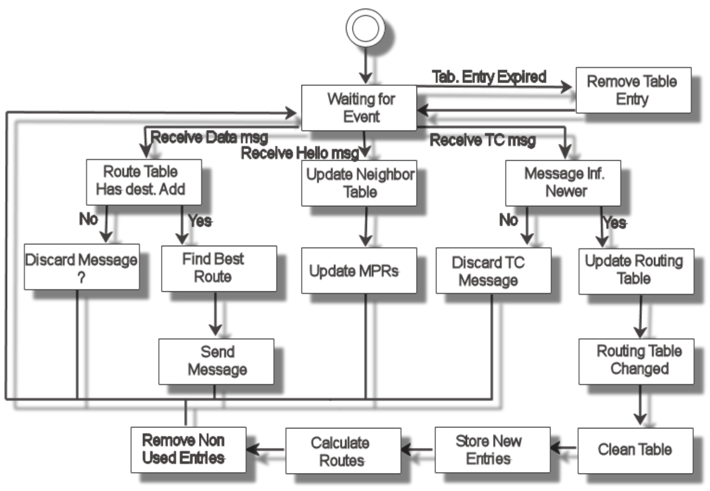

Fig. 2 - Simplified OLSR intermediate node diagram.

4.1.2 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 fond in AODV by Obradovic et al. [2] were exactly specification errors. These specification problems occur, in some extent, often because the protocol designers are much more concerned about the main protocol ideas and are somehow more careless describing the details and smaller, but important, design decisions. Fig. 2 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.

4.1.3 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 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 re-broadcast
Destination	This role does not exist; act as intermediate	This role does not exist; act as intermediate	Send msg to upper layers

Table 1 – OLSR Messages and their semantics for each node.

4.1.4 Dividing into internal and external behaviors

The main part of the verification is done by the internal model behavior. This is expected once, 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 worthy 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.

4.1.5 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 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 channel's 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.

4.1.6 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, that 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 initialized with a random value. 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.

```
#define nnodes 3 /* Interm,Origin,Dest */

chan medium = [nnodes] of  { mtype, byte, byte };
mtype = { RouteRequest, RouteReply, DataPacket };
mtype = { Origin, Destination, Intermediate };
bool packt = false;

proctype nodes()  {
    bool ttl;
    byte type;
    if /* Creates random packet */
       :: packt == false $->$  packt = true $->$
        If
            :: skip $->$ ttl= true
            :: skip $->$ ttl = false
        fi;
        if
            :: skip $->$ type = RouteRequest
            :: skip $->$ type = RouteReply
            :: skip $->$ type = DataPacket
        fi;
        medium!type(ttl); /* sends the packet */
    fi;
    if /*threat packet*/
            :: skip $->$  node = Origin
                          /*play origin node*/
            :: skip $->$  node = Destination
                          /*play dest*/
            :: skip $->$  node = Intermediate
                          /*play inter*/
    fi;
}
init {   run nodes(); }
```

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

4.1.7 Verifying the model

In SPIN a verification is done based on propositions represented in linear time temporal logic (LTL). To make the verification easier, each protocol scenario, specially 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 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:

```
[] ( !failDeliveryMessage && PathExists)
```

4.1.8 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. 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.

Other problem that occurs in OLSR 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. 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 overflow and message lost situations occurs leading to inconsistent routing tables among nodes.

5 Proposals for Routing Verification

This section will present some important proposals on formal verification for routing in wireless networks, giving special attention to their strong and weak points. Fig. 3 presents a schematic representation of the relations between the techniques presented, protocols verified, main concerns and tools used by the proposals.

In [2] and [14] Obradovic et al. show how to use the theorem prover HOL and the model checker SPIN to prove key properties of distance vector routing protocols. The technique is powerful and one of the most cited in the literature. The main disadvantage of this strategy is its intense user interaction [13]. HOL as semi automatic theorem proving tool, needs the user to guide it. Another problem is the complexity in defining the theorems and lemmas to perform the real proof.

In [12], Wibling et al. use the model checking technique to verify the Lightweight Underlay Network Ad-hoc Routing (LUNAR) Protocol. They use SPIN to verify the data and control aspects of the LUNAR protocol and the UPPAAL tool to verify the protocol timing properties. A possible drawback is that the authors only verify LUNAR, which was designed by the same group. Furthermore, the work is based on some strong assumptions: only bidirectional links are allowed, messages must be delivered in order, and each node in the network can only receive and handle one message at a time. Such assumptions, in some cases, may even prevent the whole protocol verification, if it is based on any of these points.

Chiyangwa and Kwiatkowska [3] focus their work on the timing aspects of AODV using UPPAAL. They build a timed-automata model and evaluate the effects of the standard protocol parameters on the timing behavior of AODV. In that work, they evaluate a linear topology in which the source is node *0* and the destination is node *n-1*. All other nodes involved are sequentially placed between the source and the destination. The work focuses on this peculiar topology because it intends to evaluate timing aspects of the routing discovery problem and to find the maximum possible network diameter. The work reaches its purpose, but it only verifies timing aspects of the protocol and does not consider qualitative aspects, such as loops and other routing problems.

In [13] Yuan et al. illustrate the dynamic operations of a MANET using Coloured Petri Nets. They show a simple way to model the dynamic topology changes of ad hoc networks with CPN. The great strength of the proposal is the simplicity and elegance of the model. However, because of the simplifications in the modeling process, the work does not really handle the process of sending messages. Thus, for example, there is no difference between full and incremental routing table updates. This simplification may hide important errors that are not verifiable. The technique also does not allow two different nodes to receive and process, simultaneously, broadcast messages. In this case errors caused by concurrent sending/receiving messages cannot be detected.

Renesse and Aghvami [21] 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 of SPIN is more suitable for large models. The supertrace mode of a SPIN validation can be performed in much smaller amount of memory, and still present reasonable coverage. They present simple examples in PROMELA of how to implement timers, mobility and other needful procedures. They apply their technique to the Wireless Adaptive Routing Protocol (WARP) using a five-node network. No strong justification or proof is given for using this number of nodes.

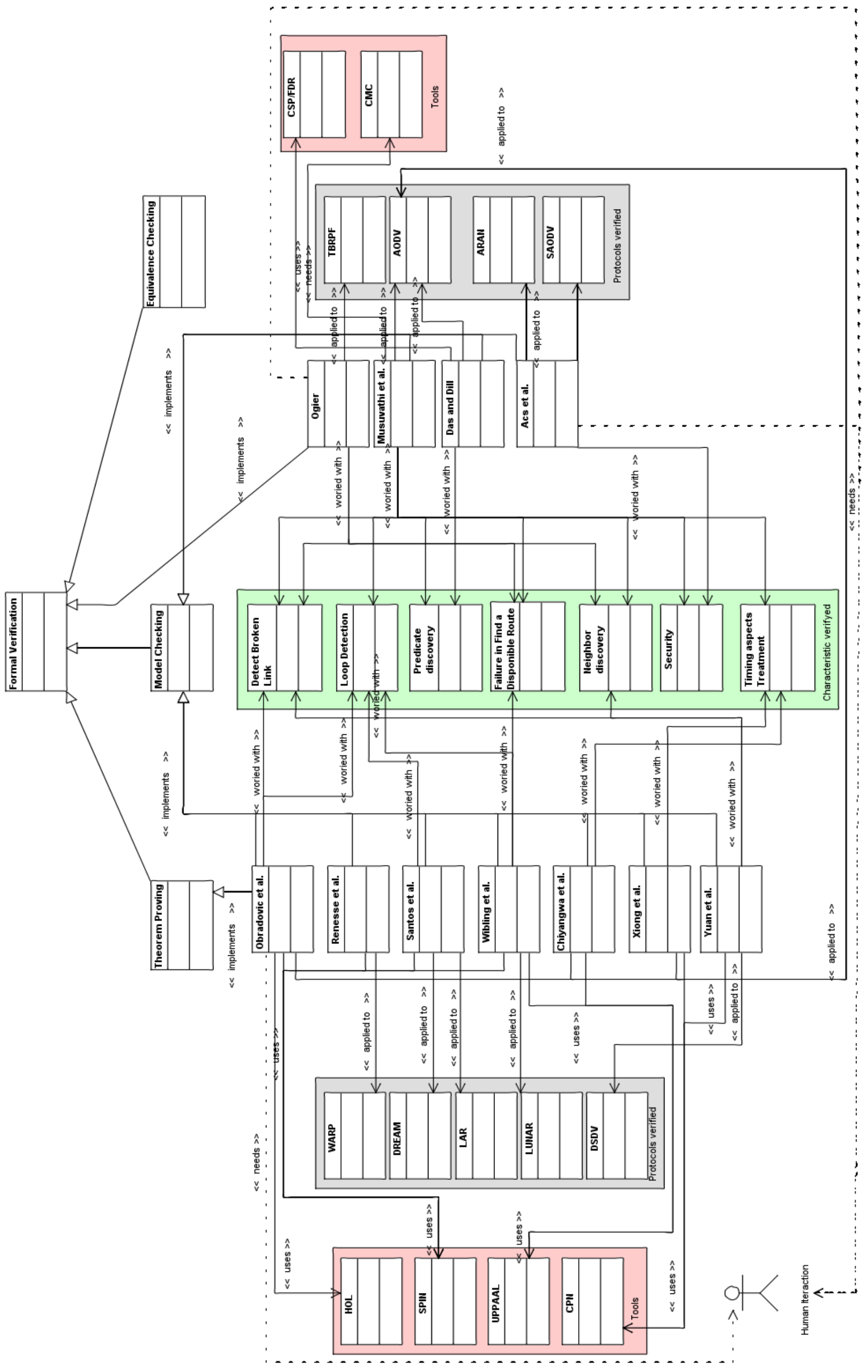

Fig. 3 - Relations among the formal verification proposals, protocols and main issues**.**

Building models to use a model checker tool is a hard and error prone task. When verifying an abstraction, rather than the code itself, it is easy to miss possible implementation errors. Observing this Musuvathi et al. [41] 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 model to abstract the system behavior. Performing the verification on the real code one 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. CMC is an interesting and promising tool, but it misses the point of evaluating profound design errors in protocols. Even the authors arguing in contrary [41], it becomes harder to verify if the protocol specification has design errors considering a real C/C++ implementation. For example Obradovic et al. [2] found a number of flaws in AODV specification exactly because they were not bounded by a real implementation. Specifically for routing protocols, another point completely missed by this technique, is the interaction among nodes. In another words, how to verify the protocol's dynamic behavior.

Zakiuddin et al. [42] propose a methodology to verify ad hoc networks 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 behaviors. They also argue that given the characteristics of the data and the tool they use, CSP and FDR, the results are applicable for an unbound number of nodes. Although the authors claim about the specification of a methodology, the proposed technique heavily relies on specific characteristics of the used tool. Another point to notice is that the application of the methodology depends on the proficiency of the designer with the tool. Once the procedure to apply the methodology is not fully specified, easily two people, applying the same technique, would arrive at different implementations and possibly results.

Xiong et al. [23] propose a timed model for AODV protocol, based on the idea of topology approximation mechanism. This mechanism describes the aggregate behavior of nodes where their long term average behaviors are of interest. With this technique the nodes and their relationships are modeled as a graph where nodes become the vertices and the links become the edges of the graph. With this, the vertex degree shows the number of neighbors of the node. This structure is then translated into colored Petri Nets. To perform the verification this work uses five nodes, but, again, there is no explanation about why to use such number. The verification also is, partially, bounded by the computational power available to the user, i.e., if there are more resources it is possible to add more nodes. This is, by no means, not a good characteristic of any technique. The protocol verification should be independent of any particular scenario.

Ács et al. [43] propose a framework model to verify security of on demand routing algorithms. Basically the authors propose the creation of two distinct models, a real world and an ideal world model. The real-world model should describe the real operation of the system, and an ideal-world model should capture what the system wants to achieve in terms of security. Then, in order to prove the security of the system, the outputs of these two models must be indistinguishable [43]. The ideal world is secure by construction. It is what one wants to achieve, so no attack can be successful on it. On the other hand the attacks can be successful at the other model, once no precautions are made in the sense of avoiding such attacks. The proposal has some drawbacks, but the main one is that it is still theoretical, and no automated proof is presented.

Das and Dill [44] 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 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 the presented technique, Predicate Abstraction, is an efficient way of reducing infinite state systems into more tractable finite state systems.

As a manual verification we can refer to the work of Ogier [45] that proves the correctness of the Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) routing protocol. Since TBRPF consists of two modules, the routing module and the neighbor discovery module, the work presents the correctness proof for both modules separately. Even though this kind of proof is not easy its results and procedures stands for TBRPF and only for it. Another point to observe is that when verifying a protocol, all cases must be considered and doing so manually it can be even harder for other protocols.

6 Directions for Future Research

The formal verification technique applied to routing algorithms for wireless networks is a quite unexplored field yet, and therefore there are lots of opportunities for new research. Indeed, the field is 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 routing 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 routing 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. [41], 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 target 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 be define a list of general situations and limits one can use this kind of technique.

The mixing of the existent tools and approaches, as 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. [43] points this need. The verification of security aspects of routing protocols for wireless networks is also a promising research field.

7 Conclusions

Formal verification is a promising technique to validate algorithms for wireless networks. Differently of which someone can think the application of formal verification techniques in the development of new routing algorithms can be easy and present an expressive increase in the quality of the protocol. 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.

8 Terminologies and Keywords

- **Formal Verification** – The mathematical proof that a formal specified system has, or has not, a given property
- **Model Checking** – Technique to verify if a defined property stands against a formal specified model
- **Model** – An abstraction of the target system, normally defined in a special purpose language
- **Theorem Proving** – A formal verification technique that employs axioms, theorems and rules of inference to proof the truth of a conclusion
- **Equivalence Checking** - The process of verifying whether two implementations of the same system are identical or not

- **Reachability** – The analysis of which states the system can reach in the next steps, giving the current state
- **State Explosion –** One of the main problems in the formal verification field. Reachability analysis creates an exponential number of states. The term *State Explosion* refers to the situation in which the state space storage grows exponentially with the size of the model
- **Coloured Petri Nets (CPN) –** Graphical language for specification and verification of computational systems
- **Symbolic representation** – The use of encoded structures to represent complex concepts trying, in this way, to decrease the need of computational resources
- **Ambiguity** – Situation where a definition may have more than one meaning
- **Routing** – The process of deciding which is the best node sequence to send a message through the network

References

1. O. Wibling, J. Parrow, A. Pears, Ad Hoc Routing Protocol Verification Through Broadcast Abstraction, 25th IFIP FORTE, Taiwan, 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, July 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, Nov. 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-00l-83, dtd l5 Aug 83; Library No. S225,7ll.
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. Yuan, Cong Billington, Jonathan, An Abstract Model of Routing in Mobile Ad Hoc Networks, Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 26 October 2005.
25. 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.
26. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled,. Model Checking, MIT Press, 1999.
27. 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.
28. Patrice Godefroid, An Approach to the State-Explosion Problem, PhD. thesis, University of Liege, Computer Science Department, 1994.
29. 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.
30. 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.
31. 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.
32. 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.
33. 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.
34. 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.
35. T. Clausen, P. Jacquet, Optimized Link State Routing Protocol (OLSR), Request for Comments: 3626, October 2003.
36. 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.
37. 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.
38. T. Murata, Petri Nets: Properties, Analysis and Applications Proceedings of the IEEE, pp. 541-580, Vol. 77, No 4, April, 1989.

39. K. Jensen, Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts, Monographs in Theoretical Computer Science, Springer-Verlag, 1997.
40. 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.
41. 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.
42. 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.
43. 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
44. Satyaki Das and David L. Dill. Counter-Example Based, Predicate Discovery in Predicate Abstraction. Formal Methods in Computer-Aided Design, Portland, Oregon, November, 2002.
45. Richard Ogier. Topology dissemination based on reverse-path forwarding (TBRPF): Correctness and simulation evaluation, Technical report, SRI International, October 2003.

9 Questions

1. What are the main purposes of using formal verification to validate algorithms for wireless networks?
   R.: Increase the quality of the protocols and avoid design flows that can compromise the algorithm performance and correctness.
2. Explain, with your own words and based on other references, the most fundamental limitation of the formal verification technique.
   R.: In opposite to some people's beliefs formal verification is not a silver bullet that can solve all the problems, it is fallible and this is its most fundamental limitation. Formal verification can only answer about properties we can think of, and more than that, for example, if your model is wrong or too simplified there is no guarantees you will reach the right results.
3. Why problems such as hidden and exposing nodes are a problem for routing algorithms?
   R.: These problems can affect deeply the way routing algorithms work and the performance of the medium access. If the routing algorithm does not take these problems into account it can degraded the whole network performance promoting collisions and requiring packet retransmissions in abundance.
4. Explain the differences among HOL, SPIN and CPN tools.
   R.: See section *Tools*.
5. When using model checking to formally verify protocols, which are the main points one should have in mind?

R.: The main point for sure is keeping the model simple but representing completely the problem. The model should contemplate all the possible relations of the real problem. Another important point is thinking hard about the properties you want to proof.

6. Search for new proposals on this field and insert them in the diagram of Fig. 3.
   R.: There is no right or wrong answer here, there are other interesting proposals on this field. Just find one and try to fit it to the figure.
7. Create a simple new routing algorithm and try to prove it is correct using SPIN.
   R.: Again there are no rights or wrongs here. The idea is to help you to get more practice with the formal verification concept. You can take as basis the verification of the section *Study Case*.
8. Find a routing algorithm for sensor or ad hoc network and, without any specific technique, try to spot three week points of it.
9. Get the same protocol and use the technique described in [4] and exemplified here and try to formally verify the protocol using SPIN.
10. Verify the protocol using HOL and CPN.