Automatized Verification of Ad Hoc Routing Protocols

Oskar Wibling, Joachim Parrow, and Arnold Pears

Department of Information Technology, Uppsala University
Box 337, SE-751 05 Uppsala, Sweden
{oskarw,joachim,arnoldp}@it.uu.se

Abstract. Numerous specialized ad hoc routing protocols are currently proposed for use, or being implemented. Few of them have been subjected to formal verification. This paper evaluates two model checking tools, SPIN and UPPAAL, using the verification of the Lightweight Underlay Network Ad hoc Routing protocol (LUNAR) as a case study. Insights are reported in terms of identifying important modeling considerations and the types of ad hoc protocol properties that can realistically be verified.

Keywords: Mobile ad hoc networks, routing protocols, formal verification, model checking, SPIN, UPPAAL, LUNAR

1 Introduction

Mobile ad hoc networks (MANETS) require efficient and correct routing protocols. So far they have mainly been evaluated by simulation and live testing, and most of the formal verifications have involved a significant amount of user intervention. We here consider a completely automatic verification strategy where the user specifies the protocol in a high level formalism and provides some general properties. These are given to a tool which will output a pass or fail answer to questions regarding key protocol properties, without involving the user in additional interaction. Compared to interactive methods much is gained in ease of use for non experts. With this intent we evaluate two model checking tools, SPIN and UPPAAL. This enables us to analyze the modeling constraints that have to be imposed, and also to provide a comparison of the tools and their suitability for the verification of ad hoc routing protocols. The evaluation additionally provides a good starting point for further work on infinite-state verification.

A MANET (Figure 1), is a spontaneous network of computers which communicate over a wireless medium. Nodes can join or leave at any time and are free to move around as they desire. There is no centralized infrastructure and so all participating nodes need to function both as end nodes and routers. Because the radio transmission range is limited, packets to distant recipients have to be routed over some intermediate node(s) in order to reach nodes outside direct transmission range. If one node has a path to another node, packets are expected to be routed there correctly.

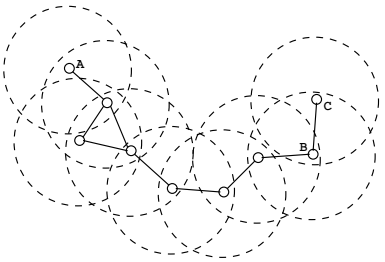

Fig. 1. A mobile ad hoc network

The situations in which ad hoc networks can or will be applied are still a topic of discussion, but scenarios such as search-and-rescue operations and sensor networks have been suggested [11, 16]. An ad hoc network needs a specifically designed routing protocol. There is ideally no pre-configuration in such a network and the network structure is expected to change constantly over time. Therefore, the nodes do not know beforehand or even during a session exactly which nodes are currently their direct neighbors. Consequently, most ad hoc routing protocols are based on broadcasting as a way to detect and map the current surroundings. There have been numerous ad hoc routing protocol proposals [28]. Currently, four of these are being considered for standardization by the Internet Engineering Task Force (IETF) MANET group [9]. They are AODV [20], DSR [12], OLSR [4] and TBRPF [19]. Very few attempts have so far been made to formally verify their operation.

As in most other computer networking areas, simulations and live testing [16] are most often employed to verify new protocols. The "Network Simulator - ns2" [10] is commonly used for simulation studies, and for real-world comparisons an assisting tool such as the "Ad hoc Protocol Evaluation (APE) testbed" [17] can be utilized. Testing and simulation are not sufficient to verify that there are no subtle errors or design flaws left in a protocol. If this goal is to be achieved approaches based on formal methods will be needed. Our emphasis is deliberately on automatic tools, since they are easier to use for non experts.

As a case study, the Lightweight Underlay Network Ad hoc Routing protocol (LUNAR) [23] has been used. LUNAR has relatively low complexity compared to many other ad hoc routing protocols and is intended as a platform to explore novel ad hoc routing strategies. Even so, it has been shown to compete well with more complex protocols such as OLSR and AODV [23]. The simplicity of the core functionality in LUNAR enables us to more clearly study the modeling of properties which are not tied to the protocol itself, such as connectivity, dynamics and broadcasting. This way we can identify key considerations that apply to the modeling of any ad hoc routing protocol.

The remainder of the paper is organized as follows. Section 2 describes ad hoc routing protocols, problems that can occur in their operation, as well as a definition of what we mean by correct operation. This section also introduces the LUNAR protocol. Section 3 describes the verification that has been performed.

Pros and cons of the different tools are discussed and lessons learned from applying them to LUNAR are presented. Section 4 covers relevant related work and finally Section 5 provides conclusions and directions for future research.

2 Ad Hoc Routing Protocols

2.1 Correct Operation

The most fundamental error in routing protocol operation (ad hoc or otherwise) is failure to route correctly. In addition to this, there are timing considerations to be taken into account, since a protocol of this kind needs to be able to react swiftly to topology changes.

In the following, when we say that a path exists between two nodes, we mean that the path is valid for some time longer than what is required to complete the route formation process. A route formation process is the process at the end of which a particular routing protocol has managed to set up a route from a source node to a destination node, possibly traversing one or more intermediate nodes. The route can thereafter be used to send packets from source to destination until the path becomes invalid as the result of a link breakage. This can occur because a node moves out of range or because the protocol itself proactively dismantles routes after a given time interval. For simplicity intermittent transmission errors at the link/physical layer are treated as link breakages in our model.

A *routing loop* in a protocol is a situation in which, somewhere along the path from the source to its destination a packet can enter a forwarding circle. This is very undesirable since there appears to be a valid path, but in reality it cannot be used to forward packets to the intended destination. As a practical example consider the description of routing loop formation in the original formulation of AODV [21] as described by Obradovic [18] (see Example 1).

Example 1. Looping behavior in AODV. The situation is depicted in Figure 2 and a brief explanation of the scenario is the following:

1. Node A initially has a route to node C through node B. The link between B and C then suddenly goes down.

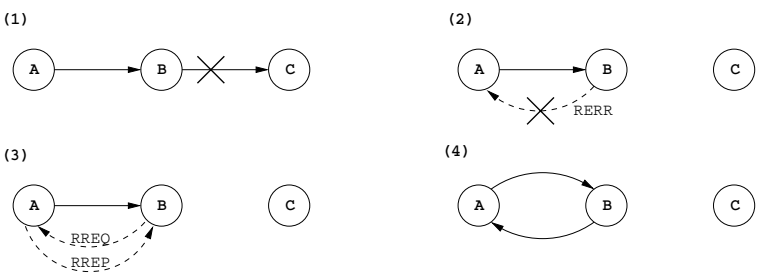

Fig. 2. Example AODV looping scenario

2. The RERR message (an inherent part of the AODV protocol) from node B to node A is lost and hence node A is not notified that the route to C has become invalid.
3. Node B then sends out a route request for node C. Node A, still thinking that it has a valid route responds that packets for C can therefore be sent to it.
4. The result is a routing loop in which A will send packets for C to node B. B on the other hand will send packets for C to node A.

These types of errors can be very subtle, and even expert designers may not be capable of detecting flaws in new protocol specifications.

To assist us in determining the correctness of a particular protocol we provide the following definition.

Definition 1. Correct operation of 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.

The definition says nothing about the behavior of the protocol when there are no paths between the nodes, but note that it excludes the possibility of loops when valid paths are available. Consider the scenario above in situation 4. If the link between nodes B and C goes up again then there is a valid path between A and C, but the protocol will keep using the loop between A and B, thus breaking the definition of correctness.

2.2 LUNAR – A Protocol Overview

Lightweight Underlay Network Ad hoc Routing (LUNAR) [23] is a reactive ad hoc routing protocol. The term "reactive" is used to denote that the protocol discovers paths only when required. However, route maintenance is proactive meaning that LUNAR rebuilds active paths from scratch every three seconds.

LUNAR creates an Internet Protocol (IP) subnet illusion by placing itself below the IP layer and above the link layer, i.e. at "layer 2.5". The IP layer of the platform on which LUNAR is running is not aware of the presence of LUNAR. Outgoing Address Resolution Protocol (ARP) solicit requests are trapped by LUNAR at which point its own multi-hop route request procedure is initiated. When a route reply has been received, the ARP table of the host is manipulated to contain an IP→selector mapping instead of an IP→Medium Access Control (MAC) address mapping. Selectors are addressing units analogous to the notion of a port and are used by LUNAR to determine the correct operation to perform on a given packet. Hence, when an outgoing IP packet is trapped, LUNAR uses the selector to determine the path for the packet. The packet is thereafter wrapped in a so called SAPF (Simple Active Packet Format) packet and delivered to its destination. When a SAPF packet is received by a node, the selector value which it contains is used to determine if it has reached its final destination, in

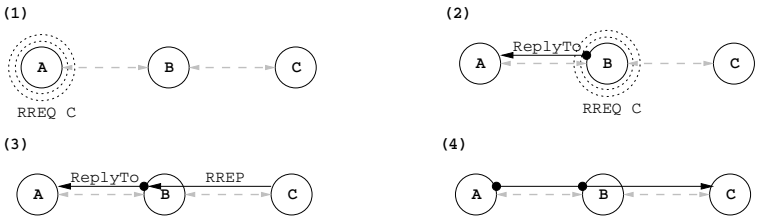

Fig. 3. LUNAR route formation overview

which case it is delivered up the IP stack. If this is not its final destination, it is forwarded along the next hop.

Broadcast dampening is an important part of the protocol and makes sure that packets are not rebroadcast more than once, thus avoiding broadcast storms. Typical usage areas for LUNAR are spontaneous ad hoc networks and wireless ad hoc Internet access links. Example 2 gives a short informal explanation of the route formation process in order to facilitate the understanding of the algorithm. Note that the simultaneous back path creation has here been omitted as a modeling simplification.

Example 2. LUNAR route formation. The situation is depicted in Figure 3. In the figure grey, dashed, bidirectional arrows indicate connectivity. The black, solid, unidirectional arrows indicate paths that have been set up in the network. An overview of the route formation process is as follows.

1. Node A wants to set up a route to node C and therefore broadcasts a LUNAR route request (RREQ).
2. The RREQ is only received by node B who concludes that it is not the node sought and therefore rebroadcasts the request. Before doing so, however, it connects the new request to a back path back to the originally requesting node (which is A).
3. The RREQ is now received by both A and C. A, through the use of the dampening mechanism concludes that the request originates from itself and drops it. C on the other hand, receives and handles the request. Since it is itself the requested node it sets up a "port" for later reception of IP packets, after which it generates a route reply (RREP) destined for B. When B receives this RREP it notes that it is a response to an original request from A, therefore it forwards the response to A. Before doing so, however, it sets up a relay so that packages received for C are forwarded to the port set up there.
4. When A receives the RREP it constructs an outgoing port to which IP packets destined for C are to be sent. This port is connected to the relay at B which is in turn connected to the port at C.

3 Case Study: Verification of LUNAR

3.1 The Model and Its Limitations

We have used a somewhat simplified version of LUNAR for our verification [27]. Apart from the simultaneous back path creation, features [23] such as automatic address configuration and forced route rediscovery, etc. are missing in our description. The same goes for the RREQ Time To Live (TTL) field since the diameters of the networks studied are small enough that we do not need to limit them explicitly.

3.2 Central Modeling Issues

When modeling an ad hoc network protocol, apart from the usual considerations regarding limiting the resulting state space, the following questions are central:

- How do we model broadcast?
- How do we model connectivity? This influences the handling of broadcast.
- How do we model topology changes (dynamics)? This directly influences the connectivity graph.

In the two sections that follow, besides giving our model checking results, we describe our solutions to each of the above issues.

3.3 Verification Using SPIN

SPIN [7] is a model checking tool that can be used for formal verification of distributed software systems. System descriptions are given in the high level language PROMELA (PROcess MEta LAnguage) and requirements to be verified can either be given as assertions directly in the code and/or by specifying correctness properties as Linear Temporal Logic (LTL) formulae. SPIN works on-the-fly, i.e. does not need to construct the complete state space prior to verification, instead this is done dynamically during processing. Furthermore, as a measure to cope with the state space explosion problem, SPIN includes a partial order reduction algorithm [8]. The state space explosion problem refers to the situation in which the state space generated by the model because of parallelism becomes so large that all the visited states cannot be stored. In the worst case the state space grows exponentially with the length of the program.

We use SPIN because of its relatively low learning threshold and powerful model checking capabilities. A PROMELA model of LUNAR has thus been constructed. The model consists of about 250 lines of code excluding comments. Our initial approach, and the one described in this work, has been to naively model the complete system and model check it as a whole. We feel that demonstrating that this is possible will lower the resistance to using these tools and increase the chances of more people verifying their protocols.

Connectivity and Dynamics. The communication "ports" where each node can be reached are modeled using PROMELA channels stored in an array indexed by node id. Node id and MAC address are used interchangeably in our model, which provides a straightforward addressing of nodes. To model connectivity a symmetric, two dimensional, array of boolean values is used. The matrix is symmetric since we assume nodes to be connected bidirectionally.

Node dynamics are modeled by modifying the connectivity matrix. In order to reduce complexity, nodes either have or do not have connectivity. No intermediate state is possible as it would be in a more realistic link/physical layer model. It would be straightforward to model lower layers in a more detailed way, but this again increases complexity and reduces the chances of successful model checking because of the state space explosion problem.

Broadcasting. Due to the transient nature of radio communication, broadcast is heavily used in most ad hoc networking protocols and LUNAR is no exception. In our model, broadcast is modeled by unicasting to all nodes with whom the sending node presently has connectivity. A PROMELA macro has been constructed for this operation. This macro consults the corresponding row in the connectivity array for the sending node and sends to all connected nodes using the channel array. The unicast operations that represent a broadcast are implemented atomically to ensure that no connectivity interruptions occur part way through the process.

Limitations Imposed. In LUNAR, both remote and local selectors are randomly selected from different ranges. Since they are specified to be 64 bits long [23], the space of possible values is huge. In our PROMELA model we are therefore forced to pick the selectors from a few limited values.

The local selectors, as their name implies, do not have to be globally unique and are therefore selected from the same range for all nodes. However, the remote selectors are meant to be globally unique and are chosen from different ranges. When a new selector is needed, the selector port value is just monotonically increased and assertions are used to make sure that the bounds are not violated. The correct bounds to use are selected by experimentation. The abstraction of selector values to a fairly limited range thus has no impact on the verification since this range is set to accommodate the needed amount of values.

The abstraction of remote selector values could have had an influence on the verification result if there was a possibility that the random selector values in an implementation of the protocol were likely to coincide. However, because of the large value space, this possibility is so minor that we consider it to be insignificant.

Channel sizes, i.e. the number of outstanding messages in a channel at one time, are in general kept as small as possible. These are selected by experimentation to hold the required number of outstanding messages and do therefore not have any impact on the verification results.

Verification. The verification of the protocol is performed by setting up the model so that one node tries to send an IP packet to another node in the network. The topology and node transitions are selected so that the two nodes are always connected, possibly by traversing one or several other nodes. Initially, no routes are set up in the network. The sending node therefore needs to initiate a route discovery process and does so by sending out a LUNAR RREQ broadcast. If and when it receives a reply it thereafter tries to send the IP packet along the route that was set up. New RREQ:s are forced after topology changes and on timeouts.

A global boolean `message_delivered` is set to `true` when the correct receiving node has received the correct IP packet from the sending node (tagged accordingly). A hop counter keeps track of the nodes traversed by the packet before it reaches its destination and an assertion is used to verify that it does not traverse more nodes than theoretically possible in the absence of loops. This assertion is checked at the receiving node prior to setting `message_delivered`.

Finally, another assertion is used in a `timeout` statement in order to check that when one node times out because of inactivity, then `message_delivered` is `true` and the message has indeed been delivered. Using SPIN we also check for the absence of deadlocks and conformance to the LTL formula `<>message_delivered` which verifies that a message will eventually be delivered.

In total, this is sufficient to show that the protocol functions correctly in each situation studied according to the statement in Definition 1. In all our scenarios we have explicitly made certain that there is a possible path between the two communicating end nodes and that each transition maintains this property. If LUNAR had any looping behavior detectable in these situations, the LTL formula above would not be fulfilled.

In our initial approach, the number of nodes is specified and then a topology is generated nondeterministically. A recursive graph traversal is then performed to see if there is a communication path possible between nodes 0 and 1. If there is not, the program ends. If there is a possible path, then the node processes are initiated and the route discovery is started, etc. In this manner, all possible topologies are considered.

However, using this approach, the state space is already large without any node connectivity transitions. When node mobility is also added, it is no longer feasible to perform an exhaustive search even for a small number of nodes. Therefore, we choose to focus on a few especially interesting scenarios which are depicted in Figure 4. These scenarios have been selected in an attempt to capture situations that could cause problems for the protocol. In scenarios (a), (c), (d), (e), (g) and (h) a situation can occur in which a route has been set up, but is then broken before the packet could be delivered. In this case, a new route should successfully be set up and the packet delivered along that one instead. In (b), (d) and (f) an extra node suddenly appears, which could potentially cause confusion for a routing protocol.

For scenarios (a), (b), (c), and (e) we are able to verify correct operation. This effectively shows that LUNAR is loop free for these topologies (and node

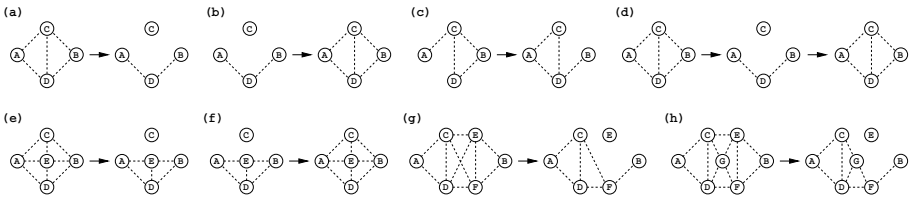

Fig. 4. Example classes of topology changes

transitions). For scenarios (d) and (f) we are not able to perform an exhaustive search because of the state space explosion (see further below). Scenarios (g) and (h) are not checked using SPIN.

Since we are doing brute force model checking it becomes important to utilize the most powerful hardware available. For this reason, we have used a Sun Fire 15k server with 36 UltraSPARC III+ CPUs at 900 MHz and 36 Gb of primary RAM to model check our scenarios. Unfortunately, SPIN models cannot currently be compiled to take full advantage of a multi-threaded architecture. There has been work on distributed LTL model checking algorithms for SPIN by Brim et al [1] and they have also implemented an experimental version. The performance increase is reported as promising. However, at this time there is no SPIN implementation with this feature publicly available.

Table 1 shows our results for scenarios (a)-(f). SPIN is here used in exhaustive search mode as opposed to the approximating bitstate hashing and hash-compact modes since we are interested in verifying correctness. Further note that both partial order reduction and COLLAPSE compression [7] are used everywhere. As a reference, the values with only partial order reduction are given within parentheses (where they differ). As can be seen, the state space rapidly grows with the number of nodes. Even using four nodes, when topology changes become just a bit more complex, the model checking fails because of memory restrictions (32 bit application). Interesting to note is that for both four and five nodes, the state space becomes much larger for the situation where one intermediate node comes up after a while, than when it goes down.

3.4 Using UPPAAL to Prove Timing Requirements

UPPAAL [15] is a tool that can be used to verify real time aspects of computer systems. The UPPAAL home page [25] provides the following definition: *"UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.)"*. The environment has a graphical user interface in which the automata are constructed by drawing them in a window on the screen. The tool contains a simulator and a verifier in which requirements are given as LTL formulae.

An UPPAAL model has been constructed in order to check timing requirements of LUNAR. The timing aspects that we focus on are to determine the-

oretical lower and upper bounds on the route formation and message delivery processes. We also check that the LUNAR model is deadlock free. In our UPPAAL model we introduce more abstraction than in the SPIN model. The main reason for this is the unavailability of more complex data structures than arrays which becomes relevant in the handling of LUNAR callbacks and redirections.

Timing Constraints. As mentioned before, the setting up of routes in an ad hoc network is usually slower than in a conventional more static network. This is because the topology needs to be rediscovered (usually by broadcasting) at regular intervals. There is a tradeoff between the exchange of control packets (used for topology analysis) and the delivery of data packets in the network. We want to achieve an optimal balance that keeps the data packet delivery times as low as possible.

Connectivity and Dynamics. As in the SPIN model, the UPPAAL model uses arrays of booleans to represent inter-node connectivity. Either, there is connectivity between two nodes or there is not. Node connectivity transitions are modeled using a separate automaton, that can at any time move to its next state whereby it manipulates the (global) connectivity table.

Broadcasting. In our version of UPPAAL broadcast channels can be used for synchronization. However, communication can not be used for value passing in UPPAAL and instead a combination of global data holders and committed locations is the recommended procedure [25]. In our LUNAR UPPAAL model broadcasting is handled similarly to unicasting except that the sending node (i.e. the broadcast initiator) has to specify that a broadcast is intended. Then, the medium automaton will set a global parameter `bc_sender` specifying the sender's identity. This is, because in the case of broadcast, the connectivity check has been deferred to the receiving nodes.

Table 1. SPIN verification results

Scenario	States generated	Transitions	All states searched	Memory used [Mb]	Time used
(a)	5715	12105	Yes	4.242 (6.188)	0.20 (0.20) s
(b)	269886	731118	Yes	33.05 (124.7)	12.33 (10.48) s
(c)	53614	128831	Yes	8.836 (30.12)	2.19 (1.92) s
(d)	4.58e+07 (8.15e+06)	1.33e+08 (2.21e+07)	No	4083 (4083)	5 h:57 min (8 min:56 s)
(e)	1.41e+06	4.59e+06	Yes	170.4 (806.6)	1:36 (1:26) min:s
(f)	3.40e+07 (7.27e+06)	1.22e+08 (2.50e+07)	No	4083 (4083)	4 h:2 min (9 min:43 s)

Limitations Imposed. The limitations in the PROMELA model are also imposed on the UPPAAL model, for the same reasons. An additional limitation that becomes relevant in the UPPAAL model is that it does not take into account computation times in the nodes. The only places in which delays occur are in the communications. This has been modeled by using a range, [`MIN_TRANSMIT_TIME`, `MAX_TRANSMIT_TIME`] of possible delays. It provides a very rough approximation of wireless network conditions and can in future versions be exchanged for a more realistic Wireless Local Area Network (WLAN) model. There are such models available [13], but we here choose the simplistic approach for lower network layers in order to reduce complexity.

In current LUNAR implementations the RREQ resending is done in an elaborate way attempting first a limited discovery using a small network radius. After this, several attempts are made with a larger radius. A timeout value is specified per "ring", i.e. per number of hops from the source node. After each RREQ timeout there is an additional waiting period before making a new attempt. In our model, however, we have chosen to settle for two timeout triggered resends. This means that in total, three route formation attempts can be made. In combination with a properly chosen timeout value, this is theoretically enough to successfully set up a route (and deliver a packet) in the scenarios studied. Our selected timeout value of 75 ms corresponds to three times the "ring" timeout in current LUNAR implementations.

Verification. The verification is performed in a manner analogous to the one described in Section 3.3. Namely, one node tries to set up a route to another node after which it attempts to send a packet there. If the route could be set up, the initiating node goes into a state `unic_rrep_rec`. If and when the correct IP packet arrives at the receiver, it goes into a state `ip_rec_ok`. Using our UPPAAL model we then verify deadlock freedom as well as timely route formation and message delivery. The LTL formulae in Table 2 are used to verify the respective properties.

There has been work done on extending UPPAAL to perform parallel model checking by Behrmann et al [2]. The advantage gained is increased speed which would in our case e.g. enable checking a wider range of scenarios. However, the total required amount of memory is larger since the state space grows when exploration is parallelized [2]. We have chosen to just study the standard (non parallel) UPPAAL distribution here.

Table 2. LTL formulae used with UPPAAL model

Property	LTL formula
Deadlock freedom	`A[] not deadlock`
Route successfully set up	`A<> Lunar0.unic_rrep_rec`
IP packet delivered	`A<> Lunar1.ip_rec_ok`

Table 3. UPPAAL verification results

Scenario	Route formation time [ms]	Message delivery time [ms]	States searched	Search completed	Memory used [Mb]	Time used
(a)	[8, 91]	[12, 99]	15072 (12789)	Yes	15.98 (16.10)	3.89 (3.23) s
(b)	[8, 16]	[12, 24]	12211 (9787)	Yes	11.42 (11.48)	2.85 (2.56) s
(c)	[8, 91]	[12, 99]	22783 (18613)	Yes	20.12 (17.28)	5.93 (5.01) s
(d)	[8, 91]	[12, 99]	50456 (41169)	Yes	37.37 (35.51)	14.91 (12.26) s
(e)	[8, 91]	[12, 99]	123196 (106257)	Yes	124.0 (109.0)	57.91 (50.44) s
(f)	[8, 16]	[12, 24]	134978 (109606)	Yes	77.39 (77.24)	47.58 (42.61) s
(g)	[12, 99]	[18, 111]	2.01e+06 (1.78e+06)	Yes	866.6 (779.4)	11:43 min:s (10:28)
(h)	-	-	2.97e+07 (2.63e+07)	No	4078 (4082)	1:59 h:min (1:50)

With the scenarios and hardware described in Section 3.3, the route formation and message delivery times in Table 3 result. UPPAAL is here used with aggressive state space reduction [2, 14]. As a reference, the values for conservative state space reduction (the default) are given within parentheses. In all our measurements the state space representation uses minimal constraint systems.

The memory and time usage in Table 3 pertains to the case where all three LTL formulae in Table 2 are checked. As communication delay we have used the range [2, 4] ms. These measurements cannot be directly compared to the ones in Table 1 for the SPIN verification because of the greater amount of abstraction introduced in the UPPAAL model. The RREQ generation strategy also differs between the models because of the absence of timing in the SPIN model. However, similar observations can be made for UPPAAL to those made in SPIN, namely that the state space grows rapidly with the number of nodes. Also, the nature of the topology changes influences the state space in a way that may sometimes be difficulty to foresee. Further note in the timing values that the shortest possible path is always the one found because of the rough link/physical layer approximation with total determinism in packet delivery.

4 Related Work

The Verinet group [26] at the University of Pennsylvania have carried out formal validation of AODV [18] and identified a flaw that could lead to loop formation. This was done using the HOL [24] theorem prover and a SPIN model of AODV in a two node setup (with an AODV router environment). They have also suggested a modification and verified that, after this, the protocol was loop free. Their approach verified the general case, but the methodology involves substantial user interaction.

Das and Dill [5] have used predicate abstraction to prove the absence of routing loops in a simplified version of AODV. The method yields a general

proof but requires human involvement in the construction of the abstraction predicates.

Engler et al [6] have studied three implementations of AODV and found 36 distinct errors, including a bug (routing loop) in the AODV specification itself. The authors used their own model checker called CMC, which checks C and C++ implementations directly, eliminating the need for a separate abstract description of the system behavior. The model checker performs its work automatically. However, prior to execution, in addition to specifying correctness properties, the user has to define an environment as well as providing guard functions for each event handler. Furthermore, their approach is not aimed at proving correctness, but rather as a method of finding bugs in the code since an exhaustive state space search can generally not be performed.

Chiyangwa and Kwiatkowska [3] have constructed an UPPAAL model of AODV in order to investigate timing properties of the protocol. To cope with the state explosion problem, a linear topology has been used with sender, receiver, and an intermediate *n_nodes* node. Using 12 intermediate nodes, the authors could conclude that the dependency of route life time on network size is undesirable and suggested a modification where it instead adapts as the network grows. This work is closely related to ours, but they have focused on UPPAAL and studied a single (linear) scenario type. The methodology involves manual consideration in constructing the specialized model. Apart from studying a different protocol, we have taken a broader view comparing two verification approaches with an emphasis on the modeling of connectivity, dynamics and broadcasting.

Xiong et al [29] have modeled AODV using colored Petri nets (CPN). To cope with the mobility problem they have proposed a topology approximation (TA) mechanism. Simulation results show that the TA mechanism can indeed simulate the mobility of a MANET without knowing its actual topology.

Theo Ruys' PhD thesis [22] discusses different methods for modeling broadcast in SPIN. An alternative to our connectivity array would be to use a matrix of channels. Furthermore, instead of a broadcast macro, a broadcast service process could have been used. Since we have utilized asynchronous channels with large enough capacity for the broadcasts, this choice has not had any impact on the asynchronous nature of the message delivery process.

5 Conclusions and Future Work

This work is to our knowledge the first to study a range of topologies in order to determine where the limit actually is when performing model checking on an ad hoc routing protocol. We demonstrate that LUNAR works correctly (according to our general definition) for a number of routing scenarios. We further provide bounds for route formation and message delivery times.

When verifying both the data and control aspects of the LUNAR protocol using SPIN and when verifying the timing properties using UPPAAL the size of network, i.e. the number of nodes involved, as well as the nature of the topological scenarios is limited due to state space storage overhead. Even if parallel model

checking approaches were used, our conclusion is that it is at this point not feasible to provide a proof for topologies of any significant size by modeling the protocol directly. On the other hand, our study enables us not only to analyze the modeling considerations that have to be imposed, but also provides us with a solid starting point for the further work we intend to pursue in the direction of infinite-state verification of ad hoc routing protocols.

Our emphasis has been on automatic model checkers in which the user provides a system specification and a number of requirements to check. The construction of models for these tools naturally still involves a certain amount of manual consideration. However, now that many of the modeling considerations have been identified, constructing verifiable models for both tools can be made rather close to the engineering activity of programming. Ultimately our goal is for the whole process to require knowledge primarily of the application, namely the ad hoc routing protocol, and not of the model checking tool. At present it is still necessary to manually (experimentally) limit the topologies and devise LTL formulae. This can be remedied by introducing specialized macros in combination with an ad hoc routing protocol preprocessor. Standard formulae could thereby be used for common situations, e.g. route setup and packet delivered.

We aim to evaluate current and upcoming parallel model checking tools in order to see where the limit in terms of number of nodes and topology dynamics currently is. We will perform these analyses on a super computer which has a very large amount of primary memory available. Further studies are also planned which involve other ad hoc routing protocols such as the ones being considered for standardization by the MANET group.

In order to provide a general proof of correctness in an automatic way, it is however not enough to study just a limited set of topologies. We need to study all available permutations for any given number of nodes. Therefore we will focus on the following directions for future research:

- Isolate the critical aspects of the ad hoc routing protocol to hand and model check those. A theorem prover can then be used to construct a general proof. This requires significant user interaction. It would be a great advantage if this process can be made more automatic.
- Employ a formal construction method rather than a post-construction verification and evaluate if there are ways to make that process more user friendly.
- Attempt an automatized infinite-state verification approach.

We currently consider the last approach as the most promising in terms of simplifying the verification process while still being able to provide a general proof.

References

[1] Jiri Barnat, Lubos Brim, and Jitka Stribrna. Distributed LTL model-checking in SPIN. Technical report, Masaryk University, December 2000. 351

[2] Gerd Behrmann, Thomas Hune, and Frits Vaandrager. Distributed timed model checking - how the search order matters. In *Proc. of 12th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Chicago, July 2000. Springer-Verlag. 353, 354

[3] Sibusisiwe Chiyangwa and Marta Kwiatkowska. Analysing timed properties of AODV with UPPAAL. Technical report, University of Birmingham, March 2004. Technical Report CSR-04-4. 355

[4] T. Clausen and P. Jacquet. Request for Comments: Optimized link state routing protocol (OLSR). `http://www.ietf.org/rfc/rfc3626.txt`, October 2003. 344

[5] Satyaki Das and David L. Dill. Counter-example based predicate discovery in predicate abstraction. In *Formal Methods in Computer-Aided Design*. Springer-Verlag, November 2002. 354

[6] Dawson Engler and Madanlal Musuvathi. Static analysis versus software model checking for bug finding. In *Proc. Verification, Model Checking, and Abstract Interpretation, 5th International Conference*, Lecture Notes in Computer Science, pages 191–210. Springer-Verlag, 2004. 355

[7] G.J. Holzmann. *The Spin Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2003. 348, 351

[8] G.J. Holzmann and D. Peled. An improvement in formal verification. In *Proc. FORTE Conference*, 1994. 348

[9] IETF MANET Working Group. MANET charter. `http://www.ietf.org/html.charters/manet-charter.html`, 2004. 344

[10] Information Sciences Institute. The network simulator – ns-2 home page. `http://www.isi.edu/nsnam/ns`, 2004. 344

[11] David B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, pages 158–163, December 1994. 344

[12] David B. Johnson, David A. Maltz, and Yih-Chun Hu. Internet draft: The dynamic source routing protocol for mobile ad hoc networks (DSR). `http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-09.txt`, April 2003. 344

[13] Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In *Proc. Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 169–187, July 2002. 353

[14] Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS)*, pages 14–24, December 1997. 354

[15] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997. 351

[16] Henrik Lundgren. *Implementation and Real-world Evaluation of Routing Protocols for Wireless Ad hoc Networks*. Licentiate thesis, Uppsala University, 2002. 344

[17] Henrik Lundgren, David Lundberg, Johan Nielsen, Erik Nordström, and Christian Tschudin. A large-scale testbed for reproducible ad hoc protocol evaluations. In *Proc. 3rd annual IEEE Wireless Communications and Networking Conference (WCNC)*, pages 412–418. IEEE, March 2002. 344

[18] Davor Obradovic. *Formal Analysis of Routing Protocols*. Phd thesis, University of Pennsylvania, 2002. 345, 354

[19] R. Ogier, F. Templin, and M. Lewis. Internet draft: Topology dissemination based on reverse-path forwarding (TBRPF). `http://www.ietf.org/internet-drafts/draft-ietf-manet-tbrpf-11.txt`, October 2003. 344

[20] C. Perkins, E. Belding-Royer, and S. Das. Request for Comments: Ad hoc on-demand distance vector (AODV) routing. `http://www.ietf.org/rfc/rfc3561.txt`, July 2003. 344

[21] Charles E. Perkins and Elizabeth M. Royer. Ad hoc on-demand distance vector routing. In *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999. 345

[22] Theo C. Ruys. *Towards Effective Model Checking*. Phd thesis, University of Twente, March 2001. 355

[23] C. Tschudin, R. Gold, O. Rensfelt, and O. Wibling. LUNAR: a lightweight underlay network ad-hoc routing protocol and implementation. In *Proc. Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN)*, February 2004. 344, 346, 348, 349

[24] University of Cambridge Computer Laboratory. Automated reasoning group HOL page. `http://www.cl.cam.ac.uk/Research/HVG/HOL/`, 2004. 354

[25] Uppsala University and Aalborg University. UPPAAL home page. `http://www.uppaal.com`, 2004. 351, 352

[26] Verinet group. Verinet home page. `http://www.cis.upenn.edu/verinet`, 2004. 354

[27] Oskar Wibling. LUNAR pseudo code description. `http://user.it.uu.se/ oskarw/lunar_pseudo_code/`, 2004. 348

[28] Wikipedia. Ad hoc protocol list. `http://en.wikipedia.org/wiki/Ad_hoc_protocol_list`, 2004. 344

[29] C. Xiong, T. Murata, and J. Tsai. Modelling and simulation of routing protocol for mobile ad hoc networks using coloured Petri nets. In *Proc. Workshop on Formal Methods Applied to Defence Systems in Formal Methods in Software Engineering and Defence Systems*, 2002. 355