

IT Licentiate theses
2005-004

Ad Hoc Routing Protocol Validation

OSKAR WIBLING

UPPSALA UNIVERSITY
Department of Information Technology





UPPSALA
UNIVERSITET

Ad Hoc Routing Protocol Validation

BY
OSKAR WIBLING

September 2005

DIVISION OF COMPUTER SYSTEMS
DEPARTMENT OF INFORMATION TECHNOLOGY
UPPSALA UNIVERSITY
UPPSALA
SWEDEN

Dissertation for the degree of Licentiate of Technology in Computer Science
at Uppsala University 2005

Ad Hoc Routing Protocol Validation

Oskar Wibling

oskarw@it.uu.se

*Division of Computer Systems
Department of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden*

<http://www.it.uu.se/>

© Oskar Wibling 2005
ISSN 1404-5117

Printed by the Department of Information Technology, Uppsala University, Sweden

Abstract

We explore and evaluate methods for validation of ad hoc routing protocols which are used to set up forwarding paths in spontaneous networks of mobile devices. The focus is automatic formal verification but we also make an initial account of a protocol performance comparison using structured live testing. State explosion is a major problem in algorithmic verification of systems with concurrently executing components. We comment on some of the best reduction methods and their applicability to our work. For reactively operating ad hoc routing protocols we provide a broadcast abstraction which enables us to prove correct operation of the Lightweight Underlay Network Ad hoc Routing protocol (LUNAR) in scenarios of realistic size. Our live testing effort has thus far uncovered significant problems in the inter-operation between TCP and ad hoc routing protocols. Severe performance degradation results in networks of just a few nodes when very little mobility is introduced. This indicates that verification for smaller network sizes is also interesting since those configurations may be the only useful ones for certain types of applications.

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

Acknowledgments

First I would like to thank my supervisors Joachim Parrow and Arnold Pears for taking me on as a PhD student and showing firm belief in me along the way thus far. You have provided valuable feedback and most importantly been there to guide me when the path ahead was far from clear.

Acknowledgments are also extended to the faculty, fellow PhD students, and staff at the IT department, in particular to past and present members of the Mobility and the Communications Research (CoRe) groups. You all contribute to making our research environment stimulating and fruitful. I would like to mention just a few colleagues, who I have worked or socialized with, by name: Jesper Bengtson, Fredrik Bjurefors, Richard Gold, Per Gunningberg, Magnus Johansson, Lisa Kaati, Daniel Lanner, Lars-Åke Larzon, Henrik Lundgren, Marcus Nilsson, Erik Nordström, Christian Rohner, Mayank Saksena, Christian Tschudin, and Björn Victor.

On the staff I especially thank Ulrika Andersson, Anne-Marie Nilsson, and Anneli Svensson for helping out with administrative items related to things such as teaching, travels, and parental leave. Magnus Berggren and Marina Nordholm can always be counted on for assistance in dealing with various practical issues. Thanks also to Joel Fredrikson for promptly providing systems support.

I want to thank my parents Eva and Edvard, my brother Johan and grandmother Irma for always being very supportive. Finally, I extend the deepest gratitude to my wife Karolina and our two daughters, Elsa and Alice. Thanks for being there to make sure I do not just work but also play. I love you.

Publications by the Author

This thesis is a summary of, and extension to, the papers listed below; referred to in the text as **Paper A-D**.

- Paper A.** Christian Tschudin, Richard Gold, Olof Rensfelt, and Oskar Wibling. LUNAR - A Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation. In *Proceedings of Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN)*, St. Petersburg, Russia, February 2004.
- Paper B.** Oskar Wibling, Joachim Parrow, and Arnold Pears. Automated Verification of Ad Hoc Routing Protocols. In *Proceedings of the 24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, Madrid, Spain, September 2004.
- Paper C.** Per Gunningberg, Henrik Lundgren, Erik Nordström, Christian Rohner, Christian Tschudin, and Oskar Wibling. Experimental Performance Evaluation of Three Ad hoc Routing Protocols. *Manuscript*.
- Paper D.** Oskar Wibling, Joachim Parrow, and Arnold Pears. Ad Hoc Routing Protocol Verification Through Broadcast Abstraction. To appear in *Proceedings of the 25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, Taipei, Taiwan, October 2005.

Comments on My Participation

Paper A: In the work related to Paper A, I participated in developing the Windows version of LUNAR. I also performed initial verifications of LUNAR using SPIN; results which are included in the paper. I co-authored the sections about the Windows version and on the formal validation.

Paper B: I am the principal author of Paper B. Regarding the work behind the paper I designed the scenarios, constructed the SPIN and UPPAAL LUNAR models, formulated the requirement properties, and executed the

verifications.

Paper C: In the work related to Paper C, I participated in all parts: design of scenarios, implementation of analysis scripts, conducting experiments, and performing the analysis. I participated in the writing of the paper as a co-author.

Paper D: I am the principal author of Paper D. I came up with the idea of abstracting away from the flooding operation in order to reduce the state explosion. Furthermore, I constructed the UPPAAL LUNAR model and executed the verifications for different parameter settings.

Contents

1	Introduction	1
2	Wireless Network Operation	5
2.1	Wireless and Ad Hoc Networks	5
2.2	Ad Hoc Routing Protocols	6
2.3	Types of Packet Transfer	8
2.4	The LUNAR Protocol	9
2.5	Correctness and Performance Issues	10
3	Protocol Validation	11
3.1	Survey of Methods	11
3.2	Formal Protocol Verification	12
3.2.1	System Description Languages	12
3.2.2	Requirement Properties and Specification Languages .	13
3.2.3	Applying the Method	13
3.2.4	The State Explosion Problem and Remedies	16
4	Ad Hoc Routing Protocol Modeling and Verification	19
4.1	Modeling the Protocol and Network	19
4.2	Initial Verification Approach	20
4.3	Coping with the State Explosion	20
4.3.1	Symbolic Representation	20
4.3.2	Partial Order Reduction	21
4.3.3	Abstraction Techniques	21
4.3.4	Exploiting Symmetry	22
5	Conclusions and Future Work	25
6	Summary of Contributions	27
	Bibliography	29
	Paper A	35

Paper B	43
Paper C	61
Paper D	71

Chapter 1

Introduction

Today wireless computing is a natural component in many people's homes while just a few years ago it was an exclusively academic and industrial phenomenon. The times when wires were used to hook up a personal computer are rapidly fading into history. Capabilities of mobile phones are reaching the point where they interact with laptop computers in wireless local area networks. The work presented in this thesis contributes to making wireless networks more reliable by making sure that they function as expected.

It makes sense to improve on the availability of mobile computing since it is becoming the norm. People are likely to expect their portable devices to exchange files at the click of a button, regardless of hardware and without prior configuration. In cities, so called "hot spots" are emerging in coffee shops and other highly frequented places. These are connection points using the standard technology for wireless networks, the base station. The radio communication range of base stations is limited, which means that users must remain in close proximity to stay connected. An interesting question concerns what can be done in the absence of such a wireless infrastructure. This might be the case when walking around in the forest or in a city park. Fortunately, mobile devices can then connect directly to each other. When obstacles are in the way or devices are out of reach it is sometimes possible to communicate using a third device as an intermediary (see Figure 1.1). This requires an additional piece of software, an *ad hoc routing protocol*.

Since the idea of ad hoc routing was first conceived, a plethora of protocols has emerged, each tailored for a particular situation. The specializations have addressed issues such as how to save battery power on limited devices, how to route quickly in highly mobile networks, and so on. If a particular routing protocol does not perform as intended, the user may notice it in the form of reduced quality of service. In the worst case an application might not work at all. An additional problem with a malfunctioning protocol is that the computer becomes vulnerable to attacks from malicious users.

In order to prohibit erroneous behavior, protocol designers generally sub-

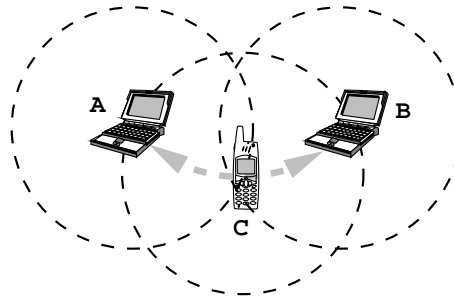


Figure 1.1: Ad hoc routing can extend communication reach. A and B cannot communicate directly because of restricted radio range, but they can communicate using C as an intermediary

ject their creations to validation. A common way to do this is to construct a software model of the protocol and simulate a large number of usage situations. The model is executed on virtual devices moving around in a simulated environment. After a given time the simulation is stopped and the result analyzed. One metric used for performance analysis is the average amount of data that reaches its destination. A second approach often used during development of the protocol is to perform live tests on actual hardware. Tests can be done more or less rigorously depending on how exactly the movement can be repeated and to what extent external factors can be controlled. In order for experiments to yield relevant information, they need to be repeatable so that a number of tests can be performed and averaged.

Neither simulation nor testing is exhaustive. They can be used to identify the bugs that are most easily found but cannot rule out protocol design errors. Still these methods can be very useful. Simulations are highly suitable for achieving a rough measure of performance in large networks and under ideal circumstances. Live testing has limitations in the size of networks that can be studied simply due to logistic reasons. Experiments need to be carefully choreographed and executed and this becomes more difficult as the number of devices increases. The advantage is in the ability to catch errors not visible in simulations due to idealizations in, for example, radio propagation models. In this thesis we report on initial results from such a structured testing study.

The third method to validate a protocol is to use formal verification methods. A protocol model is checked to see if it conforms to user requirements. Instead of executing the model, its logical structure is analyzed. This process works by using mathematical logic and can be more or less automatic. In the case of an ad hoc routing protocol the most general form of requirements can be expressed as: “The protocol should enable data packets to be forwarded between all devices in the network that are positioned on a chain of radio connectivity.” There has been previous work on formal ver-

ification of ad hoc routing protocols but the methods proposed have either been applicable only to small network configurations or required significant experience on behalf of the verifying person. The aim of our work is to develop general methods that can be applied by the average ad hoc routing protocol engineer at early stages of development to make sure that a design corresponds to the given requirements.

Chapter 2

Wireless Network Operation

2.1 Wireless and Ad Hoc Networks

The traditional wireless network uses a specific piece of hardware, the base station, as a central unit. Therefore, it is often referred to as an *infrastructure based* network. The base station is typically connected to a wired network such as the Internet and enables computers nearby to connect to it using wireless network cards. In the following we will refer to portable computers with wireless communication capabilities as mobile *nodes*. Figure 2.1 depicts a traditional wireless network with communication cells defined by the locations and ranges of base stations.

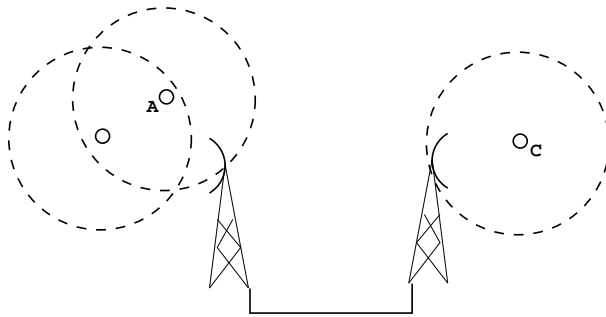


Figure 2.1: Traditional infrastructure based wireless network. Dashed circles depict radio ranges

In situations where there is no base station available or it is out of range, mobile nodes can still form a fully connected wireless network by operating in so called *ad hoc* mode [49]. Nodes within wireless transmission range of each other can then communicate directly in a peer-to-peer fashion. No relaying is done, so nodes can only communicate with their direct neighbors.

A Mobile Ad hoc Network (MANET) is a network of mobile nodes operating in ad hoc mode. A MANET has the following main characteristics:

- Nodes of the network can communicate wirelessly, for example by means of 802.11 [49] or Bluetooth [4].
- There is no fixed, centralized, infrastructure.
- Nodes are not required to remain in set positions. They can join and leave the network area at any time and change position within the network.

In short: connectivity between nodes and the network topology can vary heavily. Additionally, mobile nodes have a limited transmission range, which is one reason for a MANET to support the use of so called multi-hop paths. This means that packets can be routed over intermediate nodes in order to reach nodes outside the originating node's transmission range. Thus, all participating nodes need to function both as end nodes and routers. Other reasons for deploying multi-hop can be to save power on limited battery driven devices, to overcome obstacles, or avoid interference.

Potential applications for ad hoc networks include operations in remote areas. These can be search-and-rescue missions or disaster recovery. In everyday life one can envisage the use of ad hoc networks for quick and easy collaborative computing at conference sites, on campuses, etc. Last but not least, an important application of ad hoc networking technology is in sensor networks. There, tiny devices are spread out over an area, for example to sample environmental data which is distributed and aggregated in the network for later collection and analysis.

The so called network diameter is defined as the maximum number of hops, or intermediary nodes, between a sender and a receiver in the network. The diameter has an upper limit at which the induced delays in the ad hoc network become too high for many applications. The risk of a link break somewhere on the chain also increases with the number of hops. In **Paper A** it is argued that the maximum practical diameter, called the *ad hoc horizon* may be as short as three hops.

2.2 Ad Hoc Routing Protocols

In the basic case of an ad hoc network without multi-hop paths, a node that needs to deliver a packet to another node which is not within direct transmission range will not be able to do so. This situation can be remedied by running a routing protocol throughout the network in order to establish such paths. If there exists a multi-hop radio path between two nodes, then packets can be routed. The task of the routing protocol is to build routing tables with forwarding information. The goal is for packets to come closer to their destination and finally arrive there. Figure 2.2 depicts such a multi-hop ad hoc network.

Examples of routing protocols used in the regular Internet are the Border Gateway Protocol (BGP) [44] and Open Shortest Path First (OSPF) [37]. These protocols are not designed to react particularly swiftly to topology changes, because they do not have to. In a wired network, the time scale at which routers join or leave are rather in the range of days than seconds.

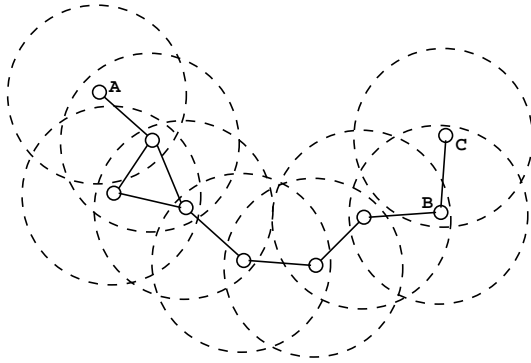


Figure 2.2: A multi-hop ad hoc network. Dashed circles depict radio ranges and solid lines represent connectivity

Because of the characteristics mentioned in section 2.1, developing a routing protocol for a mobile ad hoc network poses additional challenges in comparison to the infrastructure based case. Ad hoc routing protocols need to react quickly to topology changes and find alternative paths in order for them to be generally useful. There have been over 70 protocol suggestions [50] since the emergence of ad hoc networking. Two of these are currently up for standardization by the IETF MANET group [25], namely the Dynamic MANET On-demand Routing protocol (DYMO) [6] and version two of the Optimized Link State Routing protocol (OLSRv2) [14]. However, even though there is much effort put into developing well functioning protocols, there have been few live performance studies and even fewer formal verification attempts.

One common way to divide the proposed routing solutions is into the following main classes [19].

- Those where nodes continuously map their environment in order to already have routes available when a packet needs to be delivered to a particular destination. This follows the strategy used in wired networks and is typically done by regular distribution of control packets throughout the network. These protocols are usually referred to as proactive or table driven. Examples include the Highly Dynamic Destination-Sequenced Distance Vector routing protocol (DSDV) [41] and OLSRv2.
- Those where nodes do not take action until there is a packet that needs

to be delivered from a source to a destination. Once this happens, they attempt to generate a route as quickly as possible on-the-fly by flooding the network with a route request. These protocols are referred to as reactive or on-demand driven. Examples include DYMO, AODV [42], and DSR [28].

- Those that combine features from both of the above, commonly referred to as hybrid protocols [45]. An example in this category is the Zone Routing Protocol (ZRP) [21]. The Lightweight Underlay Network Ad hoc Routing protocol (LUNAR) (see **Paper A**) also combines reactive and proactive elements.

In the proactive and hybrid protocols there is a tradeoff between the amount of control and data traffic distributed in the network. More control traffic, that is, exchange of topology information, in general increases the reactivity to topology changes. The relation is however quite complex since data traffic throughput is dependent both on the availability of routes for the traffic and the amount of lower layer collisions with control packets. OLSR, for example, uses specialized aggregation nodes in an effort to minimize the number of packets used in the flooding of neighborhood information. Proactive protocols can to some extent be tuned for the situation in which they are intended to be used, for example if high or low mobility is expected.

The reactive protocols use a minimal amount of control traffic at times of low activity since they basically only need to set up a route to a specific node upon request. This, however, causes an inherent delay since routes may not be available beforehand. In addition, the flooding of route requests leads to bursts of control traffic. Last but not least, to detect link breaks several reactive protocol implementations use periodic broadcast beacons which add to the underlying traffic load. In general, one can say that proactive protocols are preferred in situations with little mobility and that reactive ones have a potential to become better as the amount of movement increases. In order to optimize performance, reactive protocols can use other features than control packet frequency tuning. In DSR, for example, other nodes than the intended destination can answer to a route request if they happen to be in possession of a cached route.

2.3 Types of Packet Transfer

At the link and network layers of a MANET there are in general two types of packet transfer used: unicast and broadcast. Unicast means delivering a packet to a single destination whereas a broadcast is received by all connected nodes. The difference is in reality artificial since all nodes within range can receive a transmission. In traditional networks, there is the additional possibility for a link or network level multicast operation (see, for

example, [48]). For multi-hop wireless networks a multicast or network level broadcast can be implemented by flooding [51]. We have formalized it as Propagating Localized Broadcast with Dampening (PLBD) in **Paper D** to be able to use it as a primitive operation along with unicast and broadcast.

In the DYMO protocol, the authors are using something they refer to as DYMOcast [6] which is simply a unique multicast group. A route request in DYMO is thus a message element destined for the special DYMOcast IP address. This has the effect of transmitting the packet to all connected DYMO nodes. It is not further specified how the multicast operation should be implemented. The designers have thereby chosen to separate PLBD from the ad hoc routing protocol specification in contrast to many other suggestions where it is included. The IETF MANET group is currently also working on something called Simplified Multicast Forwarding for MANET (SMF) [35]. SMF is intended to be used for efficient flooding to all nodes of a MANET routing area. Duplicate packet detection is a fundamental part of the SMF flooding process.

2.4 The LUNAR Protocol

As a case study we have used the LUNAR ad hoc routing protocol both because of its relative simplicity and because of our insight into the actual implementation. The protocol is thoroughly described in **Paper A**. In our structured testing study we have further studied implementations of the AODV and OLSR [27] protocols. Initial results from this study are accounted for in **Paper C**. To give an essence of reactive routing protocol operation in general, we here recapitulate Example 1, which is included in **Paper B**. The example informally describes how the LUNAR route formation process works. One simplification compared to protocol implementations is that the simultaneous back path creation (see **Paper A**) has been omitted.

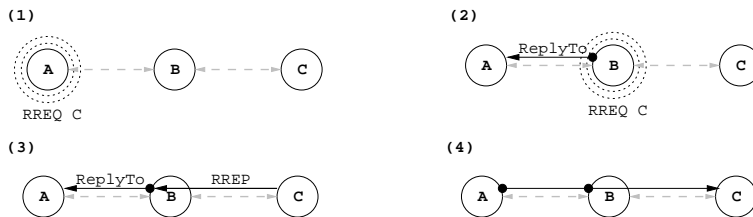


Figure 2.3: LUNAR route formation overview

Example 1 *LUNAR route formation.* The situation is depicted in Figure 2.3. In the figure grey, dashed, bidirectional arrows indicate connectivity. The black, solid, unidirectional arrows indicate paths that have been set up in the network.

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.

2.5 Correctness and Performance Issues

The main property that we would like to verify in the operation of an ad hoc routing protocol is that packets are routed correctly. In other words, if a node has direct or indirect radio connection to another node, packets are expected to be routed there. As mentioned before, an indirect connection is through other nodes, by means of multi-hop operation.

More specifically this means that a route will be found if at least one possible path exists. Additionally no routing loops are allowed to be formed. A routing loop means that a data packet can enter a forwarding circle where it does not make any progress on its way from source to destination. In **Paper B** we have included an example of such a situation that could occur in a previous version of the AODV protocol. In that paper we have also formalized the above requirements.

When the main correctness properties have been established there are also timing requirements. Routes must be found and packets delivered in a timely manner. The protocol needs to be able to react swiftly to topology changes. In **Paper B** and **Paper D** we extract time bounds in order to check the LUNAR protocol’s best and worst case behavior in the different scenarios. In **Paper C** we present initial performance results from running TCP in an ad hoc network.

Chapter 3

Protocol Validation

3.1 Survey of Methods

Computer networking protocol validation is commonly done using a combination of simulation and testing. These are both valid approaches that to some extent complement each other. Simulation offers the possibility to run a large batch of tests under identical circumstances whereby some parameter can be varied and the effect studied. A very common assisting tool, or framework, is the network simulator - ns-2 [26].

Live testing is often applied to some extent during protocol development. An important application for the method is when checking interoperability between different implementations. Live testing poses the difficulty of conducting several comparable tests, but if done in a structured way it may very well expose errors or problems not visible in simulations. The *gray zone problem*, reported by Lundgren et al. [34] is one example of such a discrepancy. In **Paper C** initial results from a structured live testing study are presented. The tool we use is called the APE testbed [38].

A third alternative is to use formal verification in order to be sure to cover all situations possible in a system model. Testing and simulation are not exhaustive methods and cannot guarantee that there are no undiscovered subtle errors or design flaws in a protocol. The objective of formal verification is to improve on reliability by reasoning about systems using mathematical logic. A formal system model can thereby be checked to fully comply with a given set of requirements.

There have been comparatively few efforts at formal verification of ad hoc routing protocols. The reason for this is twofold. First, there is the presumption that the methods are difficult to use which is to some extent true since there really is a threshold to cross before becoming proficient. The deductive methods usually require more experience before it is possible to carry out a proof for a non trivial system. Even then, it is often a very time consuming process. Second, there are inherent limitations of the methods.

In the case of deductive methods they have a potential to be very powerful and can be used to construct proofs for large or even infinite state systems. However, the proof may be notoriously difficult to find or it may not even exist because the problem is not well formulated.

Algorithmic verification methods, commonly known as model checking [9], have been more successful in terms of industrial deployment because of their easier usage. These methods have another problem that surfaces for systems composed from a set of different components that can interact in a non deterministic manner. Many possible interleavings of execution are thereby possible, leading to exponential growth of the searched state space; the state explosion problem [47].

Automatic predicate abstraction techniques, first introduced by Graf and Saïdi [20], represent a quite recent alternative which attempts to combine the advantages from theorem proving and model checking. These new techniques thus have the potential for verifying infinite state systems automatically by abstract interpretation [15] followed by, for example, symbolic model checking [36]. There is ongoing work on many fronts in order to lower the threshold of use as well as on coping with the state explosion problem. Here, we concentrate on some of the more user friendly tools, namely automatic model checkers. Our hope is to advocate the use of formal verification by the average protocol designer.

3.2 Formal Protocol Verification

3.2.1 System Description Languages

In order to verify a particular protocol it first has to be described in a structured and unambiguous way. For this, there are two main choices. Either, one can write an implementation in a regular programming language such as C and thereafter verify the code directly. This approach has been used by Engler and Musuvathi [18] to find errors in different AODV implementations. It is most often not used for exhaustive verification but rather as a method of finding program bugs, even though Engler and Musuvathi were also able to identify a routing loop error in the AODV specification.

The second approach to describing the protocol is to use a formal description language. This can either be a subset of first order predicate logic or some more high level formalism such as PROMELA (PROcess MEta LAnguage) used in the SPIN [23] tool. In reality, these languages are just representations of transition systems. It is essential that the formal description matches that of the real system implementation, but normally some parts have to be abstracted away from in order to make the problem feasible for verification. In the case of deductive verification the proof may otherwise be too difficult to construct and in the case of model checking the state space can easily blow up. When abstracting, one has to make sure

that the system model retains the same behavior as the implementation for the properties of interest.

3.2.2 Requirement Properties and Specification Languages

Requirements on the system are commonly expressed in a temporal logic such as LTL (Linear Temporal Logic) [43] or CTL (Computation Tree Logic) [10]. Requirement properties can be categorized as either *liveness* or *safety* properties [29]. Characteristic for a safety property is that a violation is detectable using a finite system run. It can informally be described using the sentence “something bad will never happen” provided that the property holds in all reachable system states. In contrast, a liveness property corresponds to the sentence “something good will eventually happen”. In order to produce a counter example for a liveness property it is sometimes necessary to study infinite system runs.

An example of a liveness property is the one we used in **Paper B** and **Paper D**, expressed somewhat simplified: *under the appropriate premises, a given routing protocol will eventually find a route to a particular destination*. As an example of a safety property, we will see the non-looping condition used in [3, 16, 30]: *the given routing protocol will not yield a state where a route is set up so that packets are forwarded in a loop, never reaching their intended destination*. This property does *not* guarantee correct route setup but prevents looping behavior, a common reason for failure.

3.2.3 Applying the Method

Model Checking

There are two main advantages of model checking in comparison to deductive methods. The first one is that once the system model has been constructed and the verification properties devised, the process is completely automatic and outputs a “yes” or “no” answer. The other advantage is the possibility to generate error traces in case a property is not fulfilled by the system. This makes it possible for the user to modify the model accordingly. The main disadvantage of model checking is its limitation to *finite state systems*. It can, however, be used in hybrid infinite state verification approaches where model checking is, for example, a component in a CEGAR (Counter-Example Guided Abstraction Refinement) loop [12]. Furthermore, model checking of symbolically represented systems can be regarded as infinite state since the original system may contain an unlimited element (such as continuous time). Using model checking, one can check safety as well as liveness properties. Model checking algorithms work by exploring the state space whereby the search stops at the first violation or when the complete execution tree has been examined. Methods can be divided into explicit state and symbolic model checking depending on if the individual states or

groups (sets) of states are used to represent the state space. Below are a few examples of model checking tools. These three tools are the ones that we have mainly used in our work. We selected them because of their widespread use, continuous development, and wide acknowledgment as being powerful.

SPIN. SPIN is an on-the-fly model checker intended for verification of software systems. System descriptions are provided using the high level language PROMELA and properties can be given in several different forms, including LTL and assertions in the code. Apart from model checking, simulation can also be performed with the tool and this simplifies debugging of the system model. Counter-example traces from the verifier can thus be loaded into the simulator. In an effort to save state space overhead, SPIN implements partial order reduction and BDD-like storage techniques.

UPPAAL. With the UPPAAL [33] tool the user can perform simulations and on-the-fly verification. Diagnostic traces can be loaded into the simulator for visualization and debugging. Modeling is done through a graphic interface where systems are described using networks of automata extended with clocks and data variables. The tool is especially targeted towards time-critical applications such as communication protocols. Properties are specified using LTL and verification is symbolic using constraints to describe clock zones. The verification problem is thus reduced to manipulating and solving constraints. Optimized data structures are used for compact representation of the constraint systems.

NuSMV. NuSMV [8] is a symbolic model checker which is a reimplementa-tion and extension of McMillan’s original SMV system [36]. System descriptions are given in the NuSMV language in which transition relations are described using propositional calculus expressions. NuSMV data types are finite. In addition to booleans there are scalars and fixed sized arrays. The additional data types are, however, translated to boolean representations prior to verification. Properties can be given in LTL or CTL and the system supports BDD-based or SAT-based (bounded) model checking.

Chiyangwa and Kwiatkowska [7] have studied timing properties of AODV using UPPAAL. Their model uses a linear topology with specialized sender, receiver and intermediate nodes.

de Renesse and Aghvami [17] have used SPIN to model check the ad hoc routing protocol WARP. They use a general 5-node topology, and provide a non-exhaustive verification (using the approximating bitstate hashing mode [23]), covering 98% of the state space.

Deductive Verification

In deductive verification the goal is to prove that a conclusion, the property to be verified, can be drawn from a given set of premises, the system description. This was previously a tedious manual process which has been speeded up with the emergence of semi-automatic tools, so called theorem provers. One advantage of this method is that it can be used to prove properties of infinite state systems, for example a protocol running in a network with an unbounded number of nodes.

An invariant is an assertion that is true in all states of a system. A safety property, expressed as an invariant, can be proven using mathematical induction. First it needs to be proven that the initial system configuration implies the assertion. In the inductive step it is then checked whether all state transitions preserve the property, that is, if the assertion holds before the transition it will also hold after it. Hence, the verification does not require an explicit state space search. This avoids the state explosion problem at the cost of a more cumbersome proof process.

The manual method was used by Ogier [40] to make a proof of correctness for the TBRPF [39] protocol. He presents a proof where induction on the number of hops is used to deduce that the routing module routes optimally in terms of least number of hops. For the discovery module he further presents a proof that the neighbor information exchanged is sufficient for the functionality of the protocol.

Bhargavan et al. [3] utilized the HOL [46] theorem prover in their deduction of route validity and freedom from routing loops in AODV. They used conditions on next node pointers, sequence numbers and hop counters to form a path invariant on pairs of nodes (on the path from source to destination). Three lemmas were then verified using SPIN after which HOL was used to prove that the three lemmas imply the path invariant theorem.

Das and Dill [16] also prove absence of routing loops in a simplified version of AODV. The strategy is similar to that of Bhargavan et al., but more automated. They use predicate abstraction and can discover most of the quantified predicates automatically by analyzing spurious abstract counter-example traces, albeit with some mechanical human involvement. 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.

Lahiri and Bryant [31, 30], following the example of Das and Dill, also verified loop freedom of AODV, but using their own tool called UCLID_PA [32]. Indexed predicate abstraction was used whereas Das and Dill's verification was done using quantified predicates. Lahiri and Bryant's verification process requires manual consideration. Apart from devising the initial set of predicates, some indexing is reportedly done manually. The initial pred-

icates used are the same as in the work of Das and Dill.

3.2.4 The State Explosion Problem and Remedies

The state explosion problem in model checking refers to the situation in which the state space storage overhead grows exponentially with the size of the model. This problem occurs because of the large number of possible interleavings between processes in a reactive concurrent system. Verification may thereby fail simply because the available amount of computer memory is limited. There have been a number of suggestions for coping with the state explosion, that is, to make verification feasible for realistically sized systems. We list the major remedies below following the description by Clarke et al. [9].

Symbolic representation. Symbolic representation refers to the use of compact data structures for representing state space. For example, by encoding the transition relations of a Kripke structure as a Binary Decision Diagram (BDD) it is possible to save storage by exploiting the often inherent regularity of a hardware or software system. Constraint system representation of continuous parameters such as clock ranges, which is done in UPPAAL, is another example of a symbolic representation. In that case it would not even be possible to store all time points explicitly regardless of the amount of available memory.

Partial order reduction. Partial order reduction [24] is an optimization, for example implemented in the SPIN tool. If a group of concurrently executing processes do not exchange any data throughout their lifetime, then it does not make a difference for the end result if they are run one after the other or in parallel. This makes verification simpler since the processes can be verified in isolation. However, once processes cooperate, for example by message passing, which is certainly the case for protocol implementations, then the possible interleavings of operation have to be taken into account when verifying the system. Partial order reduction is a way of disregarding process interleavings that produce the same global state as some other interleaving. Note that the verification property also needs to be taken into account since it might introduce additional data dependencies between processes. Keeping as much as possible local to each modeled process can thus promote partial order reduction.

Compositional reasoning. This technique [2] involves decomposing the system into components which are verified separately and in isolation from the rest. Global properties can then be inferred from the composition of the components. If there are mutual dependencies between components one can still verify each component separately

under the assumption that the other components work as expected; assume-guarantee reasoning. There are both manual and automatic approaches available for compositional reasoning.

Abstraction. Abstraction [13] is always used to some extent in formal verification since the modeling step by necessity involves some simplifications. For example, in a networking protocol one often abstracts from other parts of the protocol stack such as the physical layer. This can be approximated by a possibly lossy channel. Partial completion, such as a corrupt packet not caught by error detection codes are rarely modeled since complexity would then increase enormously. In hardware verification one assumes that individual components below a certain level of detail work as expected. In software verification a common use of abstraction is to reduce large data types by replacing them with abstract values; for example, it might be sufficient for verification to only keep information about whether a number is even or odd. As mentioned above, automatic abstraction methods are available and they pose an interesting possibility to promote the wider use of formal verification.

Symmetry. Symmetry reduction [11] works by partitioning the state space into equivalence classes. A reduced model is then constructed by taking one representative from every equivalence class. The new model will often be smaller than the original one and the state transition graph is preserved. If there are replicated components in the system, such as there is in an N-party networking protocol, there is a potential to gain state space from using this type of reduction.

Clarke et al. [9] additionally list induction as a method to cope with the state explosion. What they then refer to is the formation of a so called invariant process behaviorally equivalent to an arbitrary instantiation of a parameterized model.

Note that we have excluded optimizations that are aimed at bug checking rather than making an exhaustive verification more feasible. On-the-fly verification is one example of such a method, which means that the complete state space does not need to be constructed prior to verification. The states are instead calculated as the verification progresses.

Chapter 4

Ad Hoc Routing Protocol Modeling and Verification

4.1 Modeling the Protocol and Network

In order to perform verification of an ad hoc routing protocol, we first need to construct a formal model. In addition to modeling the protocol instance running on each node, we must specify the way in which nodes are connected. There is a question of how abstract the model should be made, the tradeoff being between verification complexity and relevance of the results.

In this work we use a rather rough model of layers above and below the network layer, where routing functionality resides. Upper layers are seen as generators and destinations of Internet Protocol (IP) packets destined for a certain network address. For the lower layers (link and physical) we specify if nodes are connected or not. No intermediate state is possible. Connected nodes can send packets to each other without loss or corruption during the time of connectivity. In the UPPAAL verifications (see **Paper B** and **Paper D**), where timed automata are used for the modeling, we additionally include a nondeterministic packet delivery delay.

Regarding the modeling of network configuration, we consider the following variants.

- Explicit modeling of topology and transitions. This means that the connectivity between nodes and changes in this connectivity are explicitly expressed in the model. This is the strategy we use initially (see **Paper A** and **Paper B**).
- Parameterization, for example on the number of network nodes. In our case we parameterize on network diameter and number of link breaks (see **Paper D**).
- Universal quantification of the parameters. This provides a general

network model which encompasses all numbers of nodes and all possible topology transitions. This is the topic of our future work.

4.2 Initial Verification Approach

Our first approach is to model each protocol instance as well as the connectivity between nodes explicitly. Topology changes are also included in their exact form in the model, for example as “nodes 0 and 2 are initially connected, but at any time this connection can break.” We make sure that the dedicated sender and receiver nodes are at all times connected by some path. Lower network layers are abstracted away from by providing a channel on which packets are delivered provided there is connectivity. For simplicity, intermittent transmission errors at the link/physical layer are treated as link breakages in our model.

As described in **Paper B**, using this approach we are rather limited in the number of nodes and the types of transitions. The maximum topology that we are able to verify consists of six nodes where one can go down at any time. We are, however, able to check liveness properties on the form “route setup will eventually succeed,” etc. at the push of a button. Furthermore, we extract bounds on route setup and initial IP packet delivery times.

Even though the ad hoc horizon limits the usable size of an ad hoc network, we would like to extend our verification to handle networks larger than a few nodes and, more importantly, for general types of node transitions. For this reason we review a few of the methods for coping with the state explosion problem accounted for in Section 3.2.4.

4.3 Coping with the State Explosion

4.3.1 Symbolic Representation

To evaluate the usefulness of symbolic state space representation methods we started to model LUNAR in the NuSMV tool. The model was explicit in the sense that we adopted a network centered view with all the messages exchanged between nodes modeled using equivalent data types. However, explicit modeling turned out to be highly suboptimal due to the conversion of scalars and arrays to boolean encoding. This led us into thinking of further abstractions that could be implemented in a suitable NuSMV model. Once we had developed the new modeling strategy, accounted for below, we were no longer as constrained by state space storage. We then chose to use the UPPAAL tool for our experiments instead because it enabled us to include timing in our model in a straightforward way. The purpose of extracting time ranges was then as an extra check to validate that we had not constructed an erroneous model. As mentioned above, the UPPAAL tool also uses symbolic

Table 4.1: SPIN partial order reduction gain

Scen.	Without partial order reduction		Using partial order reduction		Reduction gain	
	States gen.	Trans.	States gen.	Trans.	States	Trans.
(a)	9653	25049	5715	12105	41%	52%
(b)	539601	1.70e+06	269886	731118	50%	57%
(c)	106621	302862	53614	128831	50%	57%
(e)	2.78e+06	1.03e+07	1.41e+06	4.59e+06	49%	55%

representation for the time ranges. In our SPIN verification we exploited a form of symbolic representation as well by using COLLAPSE compression [22]. It did not, however, in this case enable us to verify larger scenarios.

4.3.2 Partial Order Reduction

We exploited Partial Order Reduction in our SPIN verification. Table 4.1 shows a comparison of the SPIN verifications of **Paper B** performed with partial order reduction off and on respectively. We have only included the scenarios where exhaustive verification could be performed with partial order reduction turned on. The table shows that the state space needed to search in exhaustive verification decreases with 41-50% for these scenarios. The gain from using partial order reduction seems to increase with scenario complexity although we cannot be convinced by these few measurements.

Partial order reduction has not been implemented in the currently available UPPAAL versions [1].

4.3.3 Abstraction Techniques

Manual abstraction can be applied to a general class of protocols. When thinking of suitable abstractions based on our work in **Paper B** we could note that the operation responsible for most of the complexity in a LUNAR network is the flooding of route request packets. The flooding operation was in that paper explicitly included in the routing protocol model.

As mentioned in Section 2.3 we formalize the flooding operation as PLBD in **Paper D**. In that paper we show that it is sound to abstract from the flooding transmissions along all other paths than the so called *fastest path*. The reason is due to the duplicate packet suppression which makes sure that the destination and all other nodes only take action on the first request they receive. By traversing the broadcast delivery path backwards from the destination node to the source we identify what we refer to as the *unique PLBD path* which is the same as the fastest path. The effect is that we can

move the PLBD operation out of the model and assume it to be a primitive. In other words, it may be simpler to think of the PLBD primitive as a network layer that provides unique copies of each received PLBD and only performs one resend.

This does not mean that we are not handling packet collisions. They are treated as before, namely as link breakages. We are able to verify the same properties and extract time bounds, but for much more general topologies than in **Paper B**. Still, however, there is a tradeoff between network size and number of link breaks. In **Paper D** we can verify route setup at a maximum network diameter of eight hops with two link breaks. We choose to not study the effect of more link breaks since we only perform two LUNAR resends in our model. The protocol cannot be expected to guarantee successful route discovery if more link breaks occur, since all retries may then be lost. Using a modified protocol model, we would like to extend the analysis to an arbitrary number of nodes and link breaks.

One option is to construct a fair transition system model of each protocol instance and use a very general network model in which packets are collected in a pool, an unbounded set of messages. All nodes can take action based on which messages are in the pool. This is a very general topology model in which packets can be repeated infinitely many times, arrive out of order, and in which packets may arrive at an arbitrary time to their destination. It has been used by Das and Dill [16] in their verification of the AODV routing protocol, later followed by Lahiri and Bryant [31, 30]. With this network model it is not possible to prove eventuality properties on the form “a route will eventually be set up.” Invariant properties are instead used, in this case a formulation of loop freedom.

Xiong et al. [53, 52] have modeled AODV using Petri nets. A topology approximation mechanism describes dynamic topology changes. They report on a looping situation found during a state space search of a general ten node topology. Their broadcast model works by sending out unicast messages to an average number of nodes. This number is based on the average degree and the total number of nodes in the graph. The resulting PLBD implementation is less abstract than ours and models redundant packet transfers between nodes not on the fastest path between the sender and receiver. In contrast to our approach link failure effects are not included in their model as they assume regular unicast transmissions to be globally receivable regardless of topology.

4.3.4 Exploiting Symmetry

In **Paper B** we exploited the network symmetry in a manual way by just studying one network configuration in each family of isomorphic topologies. In **Paper D** we moved further by representing each extended family by a parameter configuration. We have not, however, been able to come up

with a suitable network invariant. Symmetry reduction is not implemented in the standard distributions of SPIN and UPPAAL. There are prototype implementations of both tools including this feature.

Chapter 5

Conclusions and Future Work

We have examined how to prove general properties of ad hoc routing protocols automatically. A requirement has been to not demand anything from the protocol verifier but the most basic knowledge about formal methods. Therefore, we have studied algorithmic as opposed to interactive deductive verification methods. Using manual broadcast abstraction we were able to perform verification of protocol operation in realistic situations.

Applying broadcast abstraction still requires additional effort from the modeler in terms of thinking more abstractly about the model. The modeling style is packet centered, meaning that it focuses on packet transformations. In the case of our LUNAR model, all operations are packet driven in the sense that actions are only performed upon receiving a packet. The packet centered modeling has similarities to using message sequence charts, which should facilitate for modelers who are already familiar with that.

We were furthermore limited in the number of link breaks in the LUNAR model; there was a tradeoff with the network diameter. Other protocols may not be completely packet driven and then we cannot apply our abstraction so straightforwardly. In **Paper D** we explain additional steps that need to be taken to verify the DSR protocol. Finally, the method yields a parameterized model and variables have to be instantiated with some finite value. For all these reasons we are interested in studying how new methods combining theorem proving and model checking can be used in ad hoc routing protocol verification.

One such method is predicate abstraction [20]. In our future work we intend to apply this method to be able to provide a general proof of correctness in an automatic way. Das and Dill [16] used automatic predicate abstraction to prove loop freedom in a simplified version of AODV. We would like to improve the method by attempting different optimization strategies to construct a faster tool. The tool will be developed with the additional goal

of being user friendly. Furthermore, we want to examine if other properties than loop freedom can be proven or if loop freedom can be described in a more general way.

Regarding our structured testing study, we are currently in the process of finishing a complementary round of tests to further analyze the reasons for the poor TCP performance reported on in **Paper C**. Tests with simpler forms of traffic (UDP and Ping) are included as well as other basic tests aiming at explaining effects caused by the radio medium and MAC layer. All these analyses will be included in an upcoming technical report.

Chapter 6

Summary of Contributions

The main contributions accounted for in this thesis are the following.

- A survey on efforts on ad hoc routing protocol verification as well as a classification of their advantages and limitations.
- A formulation of the correctness requirements for ad hoc routing protocols. This definition has later been used by Chiyangwa and Kwiatkowska [7].
- A set of well defined and motivated test scenarios for use in structured live testing. These have been adopted by Borgia et al. [5] in their work on protocol performance evaluation.
- SPIN and UPPAAL models of the LUNAR protocol. These have been provided to several other researchers upon their request for use in further protocol verification efforts.
- The original idea of abstracting from the flooding operation in order to reduce the state explosion in algorithmic verification of ad hoc routing protocols.

Bibliography

- [1] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on Uppaal. <http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf>, November 2004.
- [2] Sergey Berezin, Sérgio Campos, and Edmund M. Clarke. Compositional reasoning in model checking. In *Compositionality: The Significant Difference: International Symposium (COMPOS'97)*, volume 1536 of *Lecture Notes in Computer Science*, pages 81–102. Springer–Verlag, September 1997.
- [3] K. Bhargavan, D. Obradovic, and C.A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM (JACM)*, 49(4):538–576, July 2002.
- [4] Bluetooth SIG. *Bluetooth Core Specification Version 2.0 + EDR*, 2004.
- [5] Eleonora Borgia, Marco Conti, Franca Delmastro, and Enrico Gregori. Experimental comparison of routing and middleware solutions for mobile ad hoc networks: Legacy vs cross-layer approach. In *Proc. ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis (E-WIND)*, pages 82–87. ACM Press, August 2005.
- [6] I. Chakeres, E. Belding-Royer, and C. Perkins. Internet draft: Dynamic MANET on-demand (DYMO) routing. <http://www.ietf.org/internet-drafts/draft-ietf-manet-dymo-02.txt>, June 2005.
- [7] Sibusisiwe Chiyangwa and Marta Kwiatkowska. A timing analysis of AODV. In *Proc. 7th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, volume 3535 of *Lecture Notes in Computer Science*, pages 306–321. Springer–Verlag, June 2005.
- [8] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource

- Tool for Symbolic Model Checking. In *Proc. 14th International Conference on Computer-Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer–Verlag, July 2002.
- [9] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.
- [10] E.M. Clarke and E.A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop*, volume 131 of *Lecture Notes in Computer Science*. Springer–Verlag, May 1981.
- [11] E.M. Clarke, E.A. Emerson, S. Jha, and A.S. Sistla. Symmetry reductions in model checking. In *Proc. 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 147–158. Springer–Verlag, June/July 1998.
- [12] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer–Verlag, July 2000.
- [13] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, September 1994.
- [14] T. Clausen. Internet draft: The optimized link-state routing protocol version 2. <http://www.ietf.org/internet-drafts/draft-clausen-manet-olsrv2-00.txt>, July 2005.
- [15] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press.
- [16] Satyaki Das and David L. Dill. Counter-example based predicate discovery in predicate abstraction. In *Proc. 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, volume 2517 of *Lecture Notes in Computer Science*, pages 19–32. Springer–Verlag, November 2002.
- [17] Ronan de Renesse and A.H. Aghvami. Formal verification of ad-hoc routing protocols using SPIN model checker. In *12th Mediterranean Electrotechnical Conference (IEEE MELECON)*, May 2004.

- [18] Dawson Engler and Madanlal Musuvathi. Static analysis versus software model checking for bug finding. In *Proc. 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'04)*, volume 2937 of *Lecture Notes in Computer Science*, pages 191–210. Springer–Verlag, 2004.
- [19] L.M. Feeney. A taxonomy for routing protocols in mobile ad hoc networks. Technical Report T99–07, Swedish Institute of Computer Science SICS, October 1999.
- [20] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proc. 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer–Verlag, June 1997.
- [21] Zygmunt J. Haas and Marc R. Pearlman. ZRP: a hybrid framework for routing in ad hoc networks. In *Ad Hoc Networking*, chapter 7, pages 221–253. Addison-Wesley, 2001.
- [22] G.J. Holzmann. State compression in SPIN: Recursive indexing and compression training runs. In *Proc. Third International SPIN Workshop*, pages 1–10, April 1997.
- [23] G.J. Holzmann. *The Spin Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2003.
- [24] G.J. Holzmann and D. Peled. An improvement in formal verification. In *Proc. 7th IFIP WG 6.1 International Conference on Formal Description Techniques (FORTE)*, pages 197–211, October 1994.
- [25] IETF MANET Working Group. MANET charter. <http://www.ietf.org/html.charters/manet-charter.html>, 2005.
- [26] Information Sciences Institute. The network simulator - ns-2 home page. <http://www.isi.edu/nsnam/ns>, 2005.
- [27] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *Proc. IEEE International Multi Topic Conference INMIC*, pages 62–68, December 2001.
- [28] David B. Johnson, David A. Maltz, and Josh Broch. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In *Ad Hoc Networking*, chapter 5, pages 139–172. Addison-Wesley, 2001.
- [29] E. Kindler. Safety and liveness properties: A survey. *Bulletin of the European Association for Theoretical Computer Science*, 53:268–272, June 1994.

- [30] Shuvendu K. Lahiri. *Unbounded System Verification Using Decision Procedure and Predicate Abstraction*. Phd thesis, Carnegie Mellon University, September 2004.
- [31] Shuvendu K. Lahiri and Randal E. Bryant. Predicate abstraction with indexed predicates. *ArXiv Computer Science e-prints*, July 2004. <http://arxiv.org/abs/cs.L0/0407006>.
- [32] Shuvendu K. Lahiri and Randal E. Bryant. UCLID_PA home page: Verification tool for UCLID models using predicate abstraction. http://www.cs.cmu.edu/~uclid/uclid_pred_abs/uclid_pa_page.html, 2005.
- [33] 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.
- [34] Henrik Lundgren, Erik Nordström, and Christian Tschudin. Coping with communication gray zones in IEEE 802.11b based ad hoc networks. In *5th ACM international workshop on Wireless mobile multimedia (WoWMoM)*, pages 49–55. ACM Press, 2002.
- [35] J. Macker and SMF Design Team. Internet draft: Simplified multicast forwarding for MANET. <http://www.ietf.org/internet-drafts/draft-ietf-manet-smf-00.txt>, June 2005.
- [36] Kenneth L. McMillan. *Symbolic Model Checking - An approach to the state space explosion problem*. Phd thesis, Carnegie Mellon University, May 1992.
- [37] J. Moy. Request for Comments: OSPF version 2, STD 54. <http://www.ietf.org/rfc/rfc2328.txt>, April 1998.
- [38] Erik Nordström, Henrik Lundgren, David Lundberg, Christian Tschudin, and Per Gunningberg. The APE testbed home page. <http://apetestbed.sourceforge.net/>, 2005.
- [39] R. Ogier, F. Templin, and M. Lewis. Request for Comments: Topology dissemination based on reverse-path forwarding (TBRPF). <http://www.ietf.org/rfc/rfc3684.txt>, February 2004.
- [40] Richard Ogier. Topology dissemination based on reverse-path forwarding (TBRPF): Correctness and simulation evaluation. Technical report, SRI International, October 2003.
- [41] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. ACM SIGCOMM Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

- [42] 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.
- [43] A. Pnueli. A temporal logic of programs. *Theoretical Computer Science*, 13:45–60, 1981.
- [44] Y. Rekhter and T. Li. Request for Comments: A border gateway protocol 4 (BGP-4). <http://www.ietf.org/rfc/rfc1771.txt>, March 1995.
- [45] Prince Samar, Marc R. Pearlman, and Zygmunt J. Haas. Hybrid routing: The pursuit of an adaptable and scalable routing framework for ad hoc networks. In *The Handbook of Ad Hoc Wireless Networks*, chapter 14. CRC Press, 2003.
- [46] University of Cambridge Computer Laboratory. Automated reasoning group HOL page. <http://www.cl.cam.ac.uk/Research/HVG/HOL/>, 2005.
- [47] Antti Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer-Verlag, 1998.
- [48] D. Waitzman, C. Partridge, and S. Deering. Request for Comments: Distance vector multicast routing protocol. <http://www.ietf.org/rfc/rfc1075.txt>, November 1988.
- [49] IEEE 802.11 WG. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE, std 802.11, 1999 edition, 1999.
- [50] Wikipedia. Ad hoc protocol list. http://en.wikipedia.org/wiki/Ad_hoc_protocol_list, 2005.
- [51] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proc. of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 194–205. ACM Press, 2002.
- [52] Chaoyue Xiong, Tadao Murata, and Jason Leigh. An approach to verifying routing protocols in mobile ad hoc networks using Petri nets. In *Proc. IEEE 6th CAS Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication*, volume 2, pages 537–540, May/June 2004.
- [53] Chaoyue Xiong, Tadao Murata, and Jeffery Tsai. Modeling and simulation of routing protocol for mobile ad hoc networks using colored Petri

nets. In *Proc. Workshop on Formal Methods Applied to Defence Systems in Formal Methods in Software Engineering and Defence Systems*, pages 145–153. Australian Computer Society, Inc., 2002.

Paper A

Christian Tschudin, Richard Gold, Olof Rensfelt, and Oskar Wibling. LUNAR - A Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation. In *Proceedings of Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN)*, St. Petersburg, Russia, February 2004.

LUNAR – A Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation

Christian Tschudin, Richard Gold, Olof Rensfelt and Oskar Wibling

Abstract— In this paper we describe LUNAR, a lean and efficient routing protocol for wireless ad hoc networks. We report on its implementation, on performance comparisons and on a formal validation result. The protocol departs in several ways from standard ad hoc routing protocols: it does not feature route repair, route caching, route maintenance or packet salvation. Nevertheless it closely matches the performance of AODV in settings inside of the “ad hoc horizon”. We discuss the philosophy of LUNAR, report on several ports including versions for Windows and a stripped down μ LUNAR capable of running inside a microcontroller and operating over infrared or Bluetooth.

Keywords— Wireless networks, ad hoc routing, routing protocol implementation, LUNAR, formal validation.

I. INTRODUCTION

AD-HOC networks are typically described as a group of mobile nodes connected by wireless links where every node is both a leaf node and a router. This flat routing environment causes many challenges which have resulted in a slew of routing protocols and research projects designed to solve the problems posed by ad hoc networks [7], [15], [10], [8].

Many ad hoc routing protocols started with a simple concept and then added “essential” features like route maintenance or packet salvation for staying ahead in the competition among protocols. While this happened at the level of protocol design and simulations, the availability of implementations of these refined ad hoc routing protocols is far from satisfying; Stability as well as ease of installation and configuration are other domains that usually are neglected too. This is in contrast to the continuous sophistication of ad hoc routing protocol specifications as is visible in their version numbers: All four top MANET protocol candidates went through 10–13 revision cycles, but for some protocols there is no public implementation available at all, others more or less have one reference implementation, and today none is cross-platform (Windows, Linux, Mac and PDAs).

A. The LUNAR Approach

The aim with LUNAR was to explore novel ad hoc routing strategies and to constrain ad hoc routing protocol design to pragmatic boundaries. Low protocol complexity helps to easily implement LUNAR in other environments, as we demonstrate with μ LUNAR. Lean protocols also form a good starting point for adding the important self-configuration elements, as we also show in this paper.

LUNAR adopts a hybrid routing style as it combines elements of both reactive and pro-active ad hoc routing ap-

proaches. It is reactive in the sense that it discovers paths only when required; It is pro-active in the sense that it rebuilds active paths from scratch every 3 seconds, even if everything is fine with the current path. This removes the need for additional path maintenance procedures and link repair actions and reduces the complexity of the protocol. Another design choice was the separation of delivery paths. Each host pair potentially has its own path and softstate associated with it: It is the duty of the originating node to keep that state alive. This makes implementation easier as, for example, intermediate nodes can now be fully reactive and do not have to start protocol activities by themselves.

B. Ad hoc Horizon

LUNAR limits itself to 3 hops. This decision was based on our experience (when experimenting with the standard MANET protocols over IEEE 802.11) that 3 hops is already pushing the limits in many ways. We hypothesize that there is an *ad hoc horizon* beyond which it becomes ineffective to handle topology changes as they occur in mobile wireless networks. First, when multihop routing is in place, it means that the wireless cards operate close to their limits, resulting in a highly fluctuating connectivity space: Slight position changes or objects getting in the way drastically change the neighbor set. Second, the freshness of routing information decays rapidly with the number of hops – attempts to do local repair potentially mask or at least delay the recognition of trouble spots. Third, the discovery and maintenance of routes by the use of flooding beyond the ad hoc horizon disturbs the communications of remote nodes more than it serves the local needs.

C. Underlay Routing and Internet Access

Another major departure from the MANET protocols is that LUNAR positions itself below IP. Instead of making ad hoc routes visible at the IP level, we create a subnet illusion. To the IP stack it looks as if the mobile node was connected to a LAN. Inside LUNAR we use an underlay at “layer 2.5” which emulates the LAN behavior by establishing point-to-point multihop paths or point-to-multipoint multicast trees to mimic the broadcasts at the LAN level. The underlay approach leads to immediate implementation benefits as LUNAR does not have to interact with the IP routing tables. Moreover, the underlay discovery mechanisms inside LUNAR permits to add self-configuration elements (address assignment, gateway discovery) in a very straightforward way.

II. RELATED WORK

Routing below the IP layer for ad hoc networks was independently adapted by [1] using label switching which

Christian Tschudin is a member of the Computer Networks Group, University of Basel, Switzerland. E-mail: christian.tschudin@unibas.ch. Richard Gold, Olof Rensfelt and Oskar Wibling are with the Communication Research Group of Uppsala University, Sweden. E-mail: rmg@it.uu.se, olof.rensfelt.4113@student.uu.se, oskarw@it.uu.se.

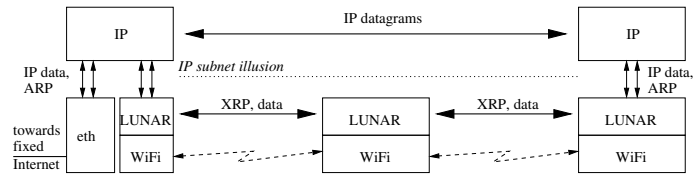


Fig. 1. LUNAR as an underlay to the IP layer

is equivalent to the “selectors” that we use in LUNAR. A similar project is [2] where the authors also aim at putting L2.5 routing logic inside the (wireless) network interface card.

There are four main routing protocols currently being considered for standardization by the IETF MANET group. Two are reactive (DSR & AODV) and two are proactive (OLSR & TBRPF). There is a DSR click router implementation available which seems stable, unfortunately it was not available at the time that this work was originally done. For AODV, which was recently raised to full RFC status, there are four full AODV implementations available. For our comparisons we used AODV-UU [14] which is a stable and mature implementation. OLSR [16] is available in a stable implementation from INRIA and appears to work well, therefore it was also included in comparisons. Unfortunately, there is no publicly available implementation of the TBRPF protocol.

For AODV, formal validations have been carried out by the Verinet group [19] at the University of Pennsylvania. Using a theorem prover and a SPIN model of AODV in a 2 node setup (with an AODV router environment), it could be shown that – with additions – it is in fact a loop free routing protocol.

III. THE LUNAR PROTOCOL

LUNAR presents itself towards the IP stack as a network interface card that attaches to a local area network. All nodes participating in the LUNAR network will appear as being one IP-hop away. Internally, this connectivity is implemented by forwarding data packets over multiple hops.

Figure 1 shows the position of the LUNAR network with respect to the traditional IP stack. The design of LUNAR is based upon the SelNet underlay network [17] which is a frugal forwarding abstraction designed to support a wide range of data forwarding and routing styles. Typically, SelNet is implemented at L2.5, but can also be put at L3.5 as an overlay where it can be used for packet redirection.

The main idea behind LUNAR is to link ad hoc path establishment to ARP. Historically, DSR also followed this approach by doing multihop ARP [9]. Typically ARP is confined to broadcasting only to the subnet that the node is currently residing in. For LUNAR, we decided to allow nodes which receive such resolution requests to rebroadcast them to reach nodes which are outside of the original radio range of the requesting node.

A. SelNet

The SelNet network basically offers a demultiplexing service. An incoming packet will be dispatched to an ap-

propriate handler routine based on a single packet header field called “selector”. As a packet is forwarded through a SelNet network, it will have a different selector on each leg of its path because the SelNet network will rewrite the combination of Ethernet address and selector on each hop, in the same way as IP forwarding rewrites the Ethernet address for each subnet crossed. A special selector value is defined for XRP packets (eXtensible Resolution Protocol) which enables to create rewriting state in remote nodes and, in the case of LUNAR, to address the LUNAR specific forwarding logic. Delivery paths dynamically obtain their own set of selector values such that XRP control traffic and the multiple ad hoc delivery paths are kept separated.

B. SelNet Control Messages (XRP)

XRP (eXtensible Resolution Protocol) is SelNet’s external data format for requests and replies. Its name stems from the philosophy that “resolution” is a mandatory step before using SelNet’s forwarding capabilities. An example would be the resolution of a neighbor’s ethernet address such that one can send data to it, or a LUNAR resolution where we want to resolve an IP address to a multihop path. In both cases SelNet will set up the necessary state inside the local host as well as in the network, and return an opaque “handle” in form of a selector value. In a second step one can use this selector for sending packets to the now resolved destination; The corresponding API consists of a simple `selnet_demux(selector, data, length); procedure`.

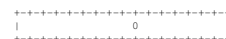
XRP messages are mere containers for parameters, its format was inspired by the CASP proposal [3] and RSVP. Here we describe the generic data structures and point to some LUNAR specific parameters. XRP messages begin with a header which is 4-Bytes long:



This header details the version, Time To Live (TTL) and the flags associated with that header. After the header, we have the various parameters that we wish to send. There can be multiple XRP parameters per packet.



A “null” object closes this list so XRP becomes a self-delimiting packet format.



A typical LUNAR “Route Request” would contain the following parameters:

- Request series (Len=12, class=request series, ctype=sel)
- Address to resolve (Len=8, class=target, ctype=IPv4)
- Requested resolution (Len=4, class=reqstyle, ctype=sel/eth)
- Reply address (Len=20, class=reply addr, ctype=sel/eth)

The request series (opaque 64 bit field in form of a selector, ctype=sel) is used to identify each discovery wave for preventing broadcast storms. We request the resolution of an IP address (address to resolve, ctype=IPv4) and wish to obtain the result in form of a selector/ethernet pair that encodes the first hop of the delivery path (requested resolution, ctype=sel/eth). The reply address is our local address where we wish to receive the result of this request.

The LUNAR “Route Reply” message will have only one parameter containing a selector and ethernet address value pair:

- Forward address (Len=4, class=reqstyle, ctype=sel/eth)

This information tells the requesting node where to send data traffic to. All data packets sent to this port will be forwarded to the stack whose IP address was given in the original route request.

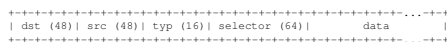
C. SelNet Packet Format

As can be seen in the XRP example above there is no field that specifies the LUNAR context in which the resolution should be carried out. In fact, this information is encoded as a “well known” selector port to which all LUNAR requests have to be sent to, whereas LUNAR replies are sent to a dynamically allocated selector port as indicated in the parameter list. This relates to SelNet’s basic demultiplexing philosophy and its one-header-field packet format:



The same selector+data format is used for a) XRP requests, b) XRP replies as well as c) pure data traffic; Differentiation is made by the selector value only.

In order to underlay IP we define a new ethernet frame type such that IP and SelNet frames can be distinguished at arrival. A complete SelNet frame over ethernet thus has the shape:



D. LUNAR Protocol Logic

LUNAR interconnects with IP by trapping all control traffic and translating it into explicit signaling at the SelNet level. When the IP stack issues an ARP request for an IP number’s Ethernet address, LUNAR translates this to a “route request” (RREQ) expressed as an XRP message. This RREQ is broadcast to all neighbor nodes. Re-broadcasting logic is then responsible for checking duplicate RREQs (each request has a unique ID which is stored on each node it arrives to), for propagating the resolution information to the next neighbors and for temporarily storing return information for the reply.

When the target node is reached, it sends back a “route reply” (RREP) – this time using a unicast message – to the

node from which it received the broadcast. Using the return information there, this reply message travels back towards the originator while at the same time creating a data delivery path towards the target node. Data traffic travels inside the delivery paths that LUNAR has established.

LUNAR also has to take care of IP broadcasts that the communication stack might be sending (e.g. broadcast pings). To this end we use a similar flooding based discovery mechanism and express such resolution requests as XRP messages too. In order to emulate a L3 broadcast we send a L2 RREQ broadcast: all neighbors match the discovery request and will rebroadcast it before replying. We delay this reply a little in order to fuse it with the replies from downstream nodes. Depending on how many downstream nodes replied, the intermediate node will install either a “broadcast forwarding handler” or a “unicast forwarding handler”. The net effect is the creation of a delivery tree where edges are using broadcast in case a node has more than 2 child nodes and using unicast otherwise.

All data forwarding state in intermediate nodes is phased out after approximately 6 seconds. Therefore, the originator of a delivery path would have to periodically refresh it. Instead of refreshing, we chose a more suitable strategy for ad hoc networks by building a *new* path every 3 seconds. This way LUNAR is able to react swiftly to topology changes, although it does not use any hello beacons or other link layer mechanisms to discover broken “links”. Old state is garbage collected by each node without additional signaling overhead.

E. Intercepting DHCP for Address Self-Configuration

We have already described how LUNAR intercepts ARP messages and turns them into RREQs for establishing forwarding paths. Similarly, LUNAR intercepts DHCP messages and translates them into its own address allocation and resource discovery strategy. In fact, each LUNAR node implements a fake DHCP server. If the DHCP client asks for a specific IP address, LUNAR will try to resolve this address using the XRP messages. If no delivery path can be set up to the given IP address, we assume that the address is not in use and LUNAR grants the corresponding lease to the DHCP client. However, if a RREP message was received, a new random IP address is picked and tested several times before handing it to the client.

Gateways are also solicited using XRP resolution messages. In a variation of LUNAR’s path setup described above, we allow several XRP replies to travel back instead of a single reply. Back at the originator we collect them and return the list of available gateways in the DHCP reply message.

IV. IMPLEMENTATION

Figure 2 shows the software decomposition of LUNAR. In a first approach we implemented LUNAR as a Linux user space program that interposes itself between the IP stack and the wireless card driver. The TUN/TAP device is used to interface with the IP stack while NETLINK provides all information on pending ARP requests in an early phase. Alternatively, one can also capture the ARP packets once the IP stack sends them out and use ARP reply

packets to signal that a delivery path was found. However, because there is no ARP packet type that could be used to *remove* an ARP entry, we still need direct access to the ARP cache. The tight interfacing with ARP is essential for throttling the IP stack as soon as a LUNAR delivery path times out. This avoids that any (TCP) packets are lost because the IP stack will know that it has to redo an ARP before continuing with the data transmission. In order to avoid a stop-and-go behavior at the ARP interface level we let LUNAR create a new delivery path after 3 seconds in parallel to the existing one and switch silently to the new. LUNAR removes an ARP entry only if no traffic is observed for a 3 second period and the path times out.

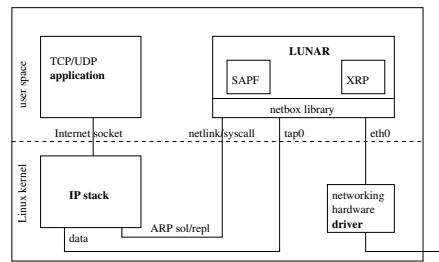


Fig. 2. LUNAR node architecture (user space implementation).

A new version of LUNAR has been development which is fully kernelized and which does not have the TUN/TAP and NETLINK dependency. This makes the deployment of the LUNAR module simpler as users do not have to recompile their kernel for TUN/TAP and NETLINK support. The kernelized LUNAR version also creates an ethernet device which is used by the IP stack like any other subnet technology. The source code is published under the GPL and can be accessed at [11].

A. Measurements

We measured the performance of LUNAR in a controlled setting of a linear network of three stationary nodes and one mobile node that roams along this linear network (see [12] for details). The following table gives the result for UDP traffic (ping and MP3 streaming):

Protocol	Ping	MP3
OLSR	89.0%	91.9%
AODV-UU	91.9%	97.9%
LUNAR	96.5%	96.8%
AODV-UU+SNR	99.1%	99.7%

LUNAR had better PING performance than plain AODV which is due to broadcast messages reaching farther than unicast messages: Because AODV trusts broadcast messages for identifying delivery paths, it can have problems in so called “communication gray zones” [12]. LUNAR is not subject to this problem because it does not really on HELLO beacons for link break detection. The patched AODV-UU+SNR version, which eliminates the gray-zone problem, now outperforms LUNAR also for the PING case. However, we consider the LUNAR performance of 96% to be sufficiently good for justifying not adding new protocol features.

B. A Tiny Multihop Ad Hoc Access Point

In order to demonstrate the reduced complexity of working below the IP layer, we fitted LUNAR and a Linux system on a single 1.4 MB floppy disk. This “LUNAR-on-a-floppy” features an automatic gateway and contains a NAT module which will map the 192.168.42.x LUNAR subnet addresses to a topologically correct IP number in the fixed Internet. As a result we have implemented a self-configuring access point which enables mobile nodes to get Internet connectivity over multiple ad hoc hops.

C. LUNAR for Embedded Systems

LUNAR’s simplicity also permitted us to implement the routing protocol in the very constrained environment of the LEGO Mindstorms RCX. The RCX is equipped with a h8300 Hitachi microcontroller with 32 KBytes of RAM and communicates over an infrared device running at 2400 Baud. The resulting system consists of a stripped down BrickOS (7KB), μ LUNAR (3.5KB), Van Jacobson header compression (4.5KB), μ IP[5] (3.5KB), a Web server and the application (1.8KB).

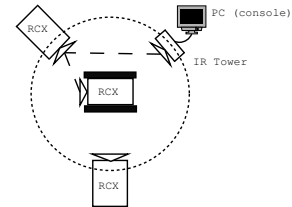


Fig. 3. Multihop IR coverage for mobile LEGO Mindstorm robot.

As an application we demoed the steering of an RCX-car over IP and the delivery of a dynamic HTML page displaying the unit’s status. Because the IR device has a limited reception angle of 60°, we covered a circle shaped area with two additional LUNAR-enabled stationary RCX nodes (see Figure 3). The mobile node in the center will always be able to communicate with the console via either a one-hop or a two-hop path.

To highlight the extremely constrained environment we note here that in μ LUNAR we merged the link layer address and the selector values, effectively making layer 2 becoming the SelNet layer and shrinking the Ethernet+selector address fields from 48+64 bits down to 16 bits.

D. LUNAR for Bluetooth Scatternets

We have developed a LUNAR version which enables us to route UDP and TCP traffic over multiple Bluetooth piconets. A Bluetooth piconet always has one master node which connects up to seven slave nodes. All communication inside a piconet passes through the master node which regularly polls the slaves. New nodes join a piconet by a lengthy *inquiry* process and will become slave nodes. In order to interconnect two piconets i.e., forming a so called scatternet, it is necessary that one node switches back and forth between the master role in one piconet and the slave role in the other.

Porting LUNAR to Bluetooth was not a straightforward task. Firstly, there is no broadcast functionality within piconets, all data communication is connection oriented. Secondly, not all connected nodes are able to inquire for new nodes, neither are all connected nodes fully discoverable. Currently we solve this problem by letting idle slaves temporarily disconnect from the master and look for new piconets to bridge to and then feeding this information back as if it was gathered via a broadcast mechanism. This proactive activity is necessary because of performance: A source node, which according to LUNAR is responsible for establishing the delivery tunnel and sending the data, would otherwise have to frequently disconnect and subsequently reconnects only for finding nodes in potential neighbor piconets.

E. LUNAR for Windows

In an effort to provide LUNAR for multiple platforms, a Windows XP/2000 version has recently been produced. It is based on the new kernelized Linux version of LUNAR and is implemented as an NDIS intermediate network driver [13]. NDIS¹ is an interface for network drivers used in the latest Windows operating systems. In this manner, Ethernet packets going out to or coming in from the network card can be intercepted and new packets injected in both directions. This is the main functionality that is required by LUNAR since it operates at a layer 2.5.

The porting process was greatly simplified due to the strict modularization of kernelized LUNAR which separates the core logic from the networking and other platform specific details. Hence, the core logic is unmodified in the Windows version and the only parts that have been changed are the LUNAR `netbox` and `platform` modules. These modules respectively contain networking and various other platform specific functionality needed by LUNAR. To this end we wrote an NDIS wrapper which offers the same (Linux kernel) functionality that the LUNAR kernel module relies on.

V. DISCUSSION

A. Forced Path Re-establishment

At first sight, the forced re-establishment of delivery paths after 3 seconds seems unusual and inefficient. However, it keeps up well with the actual behavior of other ad hoc routing protocols. It takes approximately 2 seconds for AODV to conclude that two consecutive HELLO messages from a neighbor were lost and that the topology changed, before AODV will start its route repair or rediscovery procedure. For LUNAR, the change in topology can occur anytime inside the LUNAR interval thus on average at 1.5 seconds. We were able to observe this faster reaction time in a qualitative way when listening to streaming of MP3 over AODV and LUNAR enabled ad hoc networks.

B. Formal Validation

LUNAR relies on the efficient forwarding of data packets using a simple packet format which features only a single field called the “selector”. Hence, there is no TTL field

for data packets which can prevent packets from looping, should the routing protocol configure such a looping forwarding path. One safety net consists in that all forwarding entries in a node keep a “forwarding credit”, which is decremented by every packet passing through as well as a soft state garbage collection mechanism. Nevertheless, it is important that the routing protocol does not lead to forwarding loops.

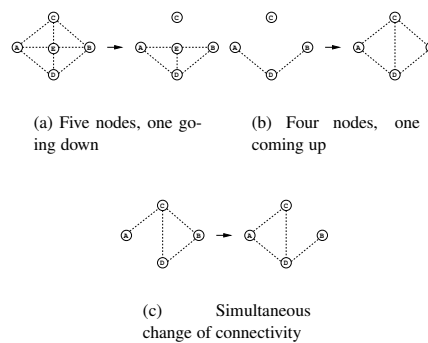


Fig. 4. Example classes of topology changes.

In order to verify the correct operation of LUNAR we have carried out formal verifications using SPIN [6]. A LUNAR version has been implemented in the modeling language PROMELA in about 450 lines of code including topology generation. We have used this model to successfully verify the correctness of the established paths by varying over the topology.

Due to the state space explosion of this exhaustive search approach, we are so far limited to verify configurations with up to 5 nodes and belonging to selected topology classes. Figure 4 shows various topology changes that were considered (single nodes coming and going away, and connectivity being simultaneously gained and lost at different places in the network).

VI. CONCLUSIONS

We have introduced the LUNAR ad hoc routing protocol which targets the common-case of network clouds with 10-15 nodes and a diameter of up to three hops. We believe that such settings will be the most popular ones where ad hoc networks can and will be put into operation. More specifically, in larger settings and for IEEE 802.11 there are such severe degradations occurring under any ad hoc routing scheme that we do not consider this to be a relevant use case that a routing protocol should try to address.

Although the LUNAR protocol does not include any route repair or packet salvation optimizations, it offers comparable performance at less than half of the code size of other MANET protocols (despite LUNAR containing self-configuration and Internet connectivity logic). LUNAR successfully exploits the constraints assumed concerning the network size, which justifies the extremely lightweight approach.

¹Network Driver Interface Specification

ACKNOWLEDGEMENTS

The authors would wish to thank Henrik Lundgren and Erik Nordström for their performance comparison. We also thank David Rosén for his work in the μ LUNAR project.

REFERENCES

- [1] Arup Acharya, Archan Misra and Sorav Bansal. "A Label-switching Packet Forwarding Architecture for Multi-hop Wireless LANs" Proc WoWMoM'02, Sep 2002, Atlanta, USA
- [2] R. Bernasconi, I. Defilippis, S. Giordano and A. Puiatti. "An Enhanced MAC Architecture for Multi-hop Wireless Networks". In Proc IFIP 6th Personal Wireless Communications PWC2003, Venice, LNCS 2775, Sep 2003.
- [3] CASP: Cross Application Signalling Protocol. Henning Schulzrinne <http://www.cs.columbia.edu/IRT/casp/>
- [4] DSR Click Router Implementation homepage Homepage: <http://pecolab.colorado.edu/DSR.html>
- [5] Adam Dunkels. "uIP - A Free Small TCP/IP Implementation for 8- and 16-bit Controllers" <http://www.dunkels.com/adam/uip/>
- [6] G.J. Holzmann. The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, 2003.
- [7] IETF MANET Working Group. MANET Charter, 2000. <http://www.ietf.org/html.charters/manet-charter.html>.
- [8] Philippe Jacquet, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Internet draft: Optimized link state routing protocol, 2001. <http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-05.txt>.
- [9] David B. Johnson. "Routing in ad hoc networks of mobile hosts". In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., 1994.
- [10] David B. Johnson, David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. "The dynamic source routing protocol for mobile ad hoc networks", 2000. <http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-04.txt>.
- [11] LUNAR home pages: <http://cn.cs.unibas.ch/projects/lunar/> and <http://www.docs.uu.se/selnet/lunar/>.
- [12] H. Lundgren, E. Nordström, C. Tschudin "Coping with Communication Gray Zones in IEEE 802.11b based Ad hoc Networks" In *WoWMoM 2002*. <http://www.docs.uu.se/docs/research/projects/scanet/aodv/grayzones.pdf>
- [13] Microsoft Corporation. "Network Devices and Protocols: Windows DDK - NDIS Intermediate Drivers". http://msdn.microsoft.com/library/en-us/network/hh/network/301int_0x2f.asp.
- [14] Erik Nordström and Henrik Lundgren. AODV-UU: AODV Implementation. <http://www.docs.uu.se/scanet/aodv>.
- [15] Charles Perkins and Elizabeth M. Royer. Internet draft: ad hoc on-demand distance vector (AODV) routing, 2000. <http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-07.txt>.
- [16] Adokoe Plakoo and Anis Laouiti. OLSR implementation. <http://menetou.inria.fr/olsr/>.
- [17] Christian Tschudin and Richard Gold. "SelNet: A Translating Underlay Network". Uppsala University Technical Report 2003-020
- [18] Christian Tschudin and Richard Gold. "Lightweight Underlay Network Ad-Hoc Routing" implementation. Uppsala University Technical Report 2003-021.
- [19] Verinet group. Home page: <http://www.cis.upenn.edu/verinet>.
- [20] Oskar Wibling. "Porting LUNAR to Windows". Communications Research Group, Technical Report. Uppsala University, 2003.

Paper B

Oskar Wibling, Joachim Parrow, and Arnold Pears. Automated Verification of Ad Hoc Routing Protocols. In *Proceedings of the 24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, Madrid, Spain, September 2004.

© 2004 Springer-Verlag. Reprinted with permission.

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.

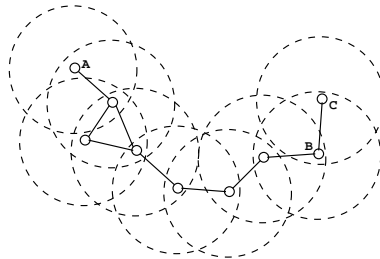


Fig. 1. A mobile ad hoc network

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 [1, 2]. 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 [3]. Currently, four of these are being considered for standardization by the Internet Engineering Task Force (IETF) MANET group [4]. They are AODV [5], DSR [6], OLSR [7] and TBRPF [8]. Very few attempts have so far been made to formally verify their operation.

As in most other computer networking areas, simulations and live testing [2] are most often employed to verify new protocols. The “Network Simulator - ns2” [9] is commonly used for simulation studies, and for real-world comparisons an assisting tool such as the “Ad hoc Protocol Evaluation (APE) testbed” [10] 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) [11] 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 [11]. 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 [12] as described by Obradovic [13] (see Example 1).

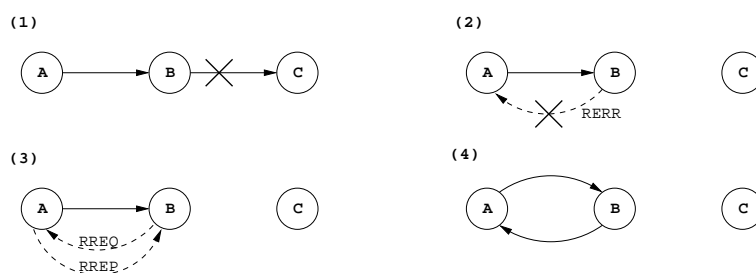


Fig. 2. Example AODV looping scenario

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.
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) [11] 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 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.

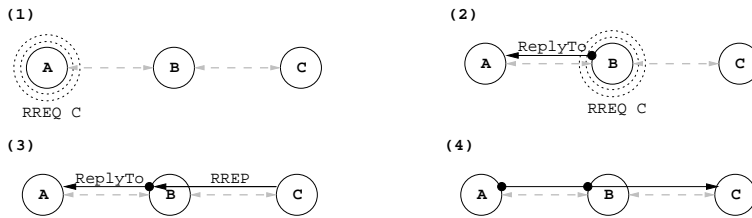


Fig. 3. LUNAR route formation overview

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 [14]. Apart from the simultaneous back path creation, features [11] 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 [15] 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 [16]. 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 [11], 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 $\langle \text{message_delivered} \rangle$ 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

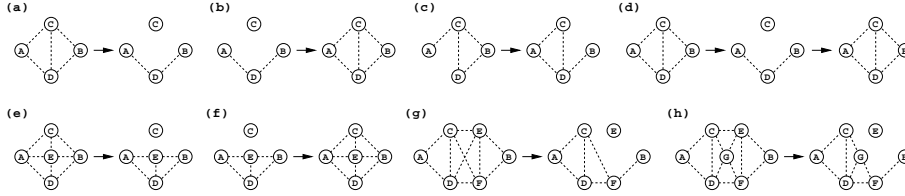


Fig. 4. Example classes of topology changes

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 [17] 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 [15] 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 [18] is a tool that can be used to verify real time aspects of computer systems. The UPPAAL home page [19] 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.

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)

An UPPAAL model has been constructed in order to check timing requirements of LUNAR. The timing aspects that we focus on are to determine theoretical 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 [19]. 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.

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 [20], 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.

Table 2. LTL formulae used with UPPAAL model

Property	LTL formula
Deadlock freedom	$A[] \text{ not deadlock}$
Route successfully set up	$A\langle\rangle \text{ Lunar0.unic_rrep_rec}$
IP packet delivered	$A\langle\rangle \text{ Lunar1.ip_rec.ok}$

There has been work done on extending UPPAAL to perform parallel model checking by Behrmann et al [21]. 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 [21]. We have chosen to just study the standard (non parallel) UPPAAL distribution here.

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 [21, 22]. 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.

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)

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 difficult 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 [23] at the University of Pennsylvania have carried out formal validation of AODV [13] 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 [25] 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 [26] 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 [27] 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 [28] 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 [29] 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. Johnson, D.B.: Routing in ad hoc networks of mobile hosts. In: Proc. IEEE Workshop on Mobile Computing Systems and Applications. (1994) 158–163

2. Lundgren, H.: Implementation and Real-world Evaluation of Routing Protocols for Wireless Ad hoc Networks. Licentiate thesis, Uppsala University (2002)
3. Wikipedia: Ad hoc protocol list. http://en.wikipedia.org/wiki/Ad_hoc_protocol_list (2004)
4. IETF MANET Working Group: MANET charter. <http://www.ietf.org/html.charters/manet-charter.html> (2004)
5. Perkins, C., Belding-Royer, E., Das, S.: Request for Comments: Ad hoc on-demand distance vector (AODV) routing. <http://www.ietf.org/rfc/rfc3561.txt> (2003)
6. Johnson, D.B., Maltz, D.A., Hu, Y.C.: 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> (2003)
7. Clausen, T., Jacquet, P.: Request for Comments: Optimized link state routing protocol (OLSR). <http://www.ietf.org/rfc/rfc3626.txt> (2003)
8. Ogier, R., Templin, F., Lewis, M.: Internet draft: Topology dissemination based on reverse-path forwarding (TBRPF). <http://www.ietf.org/internet-drafts/draft-ietf-manet-tbrpf-11.txt> (2003)
9. Information Sciences Institute: The network simulator - ns-2 home page. <http://www.isi.edu/nsnam/ns> (2004)
10. Lundgren, H., Lundberg, D., Nielsen, J., Nordström, E., Tschudin, C.: A large-scale testbed for reproducible ad hoc protocol evaluations. In: Proc. 3rd annual IEEE Wireless Communications and Networking Conference (WCNC), IEEE (2002) 412–418
11. Tschudin, C., Gold, R., Rensfelt, O., Wibling, O.: LUNAR: a lightweight underlay network ad-hoc routing protocol and implementation. In: Proc. Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN). (2004)
12. Perkins, C.E., Royer, E.M.: Ad hoc on-demand distance vector routing. In: Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications. (1999) 90–100
13. Obradovic, D.: Formal Analysis of Routing Protocols. Phd thesis, University of Pennsylvania (2002)
14. Wibling, O.: LUNAR pseudo code description. http://user.it.uu.se/~oskarw/lunar_pseudo_code/ (2004)
15. Holzmann, G.: The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, Reading, Massachusetts (2003)
16. Holzmann, G., Peled, D.: An improvement in formal verification. In: Proc. FORTE Conference. (1994)
17. Barnat, J., Brim, L., Stribrna, J.: Distributed LTL model-checking in SPIN. Technical report, Masaryk University (2000)
18. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. Int. Journal on Software Tools for Technology Transfer **1** (1997) 134–152
19. Uppsala University, Aalborg University: UPPAAL home page. <http://www.uppaal.com> (2004)
20. Kwiatkowska, M., Norman, G., Sproston, J.: 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. (2002) 169–187
21. Behrmann, G., Hune, T., Vaandrager, F.: 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, Springer-Verlag (2000)

22. Larsen, K.G., Larsson, F., Petterson, P., Yi, W.: Efficient verification of real-time systems: Compact data structures and state-space reduction. In: Proc. 18th IEEE Real-Time Systems Symposium (RTSS). (1997) 14–24
23. Verinet group: Verinet home page. <http://www.cis.upenn.edu/verinet> (2004)
24. University of Cambridge Computer Laboratory: Automated reasoning group HOL page. <http://www.cl.cam.ac.uk/Research/HVG/HOL/> (2004)
25. Das, S., Dill, D.L.: Counter-example based predicate discovery in predicate abstraction. In: Formal Methods in Computer-Aided Design, Springer-Verlag (2002)
26. Engler, D., Musuvathi, M.: 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, Springer-Verlag (2004) 191–210
27. Chiyangwa, S., Kwiatkowska, M.: Analysing timed properties of AODV with UP-PAAL. Technical report, University of Birmingham (2004) Technical Report CSR-04-4.
28. Xiong, C., Murata, T., Tsai, J.: 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)
29. Ruys, T.C.: Towards Effective Model Checking. Phd thesis, University of Twente (2001)

Paper C

Per Gunningberg, Henrik Lundgren, Erik Nordström, Christian Rohner, Christian Tschudin, and Oskar Wibling. Experimental Performance Evaluation of Three Ad hoc Routing Protocols. *Manuscript*.

Experimental Performance Evaluation of Three Ad hoc Routing Protocols

Per Gunningberg, Henrik Lundgren, Erik Nordström, Christian Rohner, Oskar Wibling
Uppsala University, Sweden
Christian Tschudin
University of Basel, Switzerland

ABSTRACT

We report on experimental performance evaluation of three ad hoc routing protocols in an indoor environment. Three scenarios with simple mobility and traffic patterns are used. In each of the experiments there is a single TCP traffic flow and no additional cross-traffic. The mobility patterns have been selected so that there exists at least one available path between source and destination at all times during a test run. Despite this we find that TCP typically stalls for at least 40% of the total test run duration and periods of up to twenty seconds without progress can be observed.

1. INTRODUCTION

A mobile ad hoc network (MANET) is a temporary network of wirelessly communicating devices. Multi-hop communication is often permitted, meaning that nodes do not need to be in direct radio contact in order to exchange packets. Intermediate nodes then function as routers, cooperatively forwarding traffic. Since nodes can move around, connectivity varies, and specialized routing protocols are needed in order to react swiftly to topology changes. Such routing protocols have been studied extensively through simulation. The effects of real-world factors on these protocols are however not yet thoroughly explored or well understood.

This paper presents results from an ongoing effort to experimentally compare ad hoc routing protocols and their performance in real world settings. The focus is on three ad hoc routing protocols that have implementations mature enough that they can say something about the underlying protocol logic. We subject the protocols to TCP traffic in three scenarios with limited mobility. The scenarios let us isolate and study the effects of communication gray zones [11] and topology changes. Gray zones are geographic locations where broadcast packets can be successfully received but unicast packets cannot. In IEEE 802.11b, the main reason why this situation occurs is the difference in transmission range between 2 Mbit/s broadcast and 11 Mbit/s unicast traffic. Since the broadcast packets are in general smaller in size this additionally contributes to their longer range. An ad hoc routing protocol may suffer from these gray zones if the protocol's neighbor sensing mechanism is based on broadcast packets. It may install routes based on broadcasts even though unicast communication is not possible. In mobile scenarios where links are lost and regained, the gray zones are likely to occur just before a link is lost and just after a link is established. Our results show that both topology changes and gray zones have a significant impact on TCP progress.

The rest of the paper is organized as follows. Related efforts are accounted for in Section 2. We describe our methodology and experimental setup in Section 3. The studied routing protocols are

briefly presented in Section 4. Section 5 describes our scenarios. In Section 6 we present and discuss our results. We conclude the paper in Section 7.

2. RELATED WORK

Toh et al. [20] have performed outdoor experiments on ad hoc networks running the Associativity-Based Routing (ABR) protocol. Using the Ping utility in a static four node linear topology, with no additional cross-traffic, they examined the impact of varying packet size, ABR beaconing interval, and hop count on route discovery time, communication throughput, end-to-end delay, and packet loss. The authors also performed simple mobility experiments in order to evaluate incurred route reconstruction time. Moving nodes were emulated by removing and exchanging of 802.11 cards in a five node network. TCP throughput results using FTP over 1-2 hop static wireless links are also presented. As hop count and total file size increase, high frequency ABR beaconing significantly increases total transfer time because of the additional wireless channel contention.

Desilva and Das [4] have also evaluated transport protocol performance in static AODV networks. Route breaks are emulated by purging of routing table entries. The authors report on poor transport protocol performance beyond two hops at moderate to high loads. Using a linear topology, maximum UDP throughput drops considerably for paths longer than two hops. TCP behaves more and more irregularly as the number of hops are increased. At four hops the achievable throughput is an order of magnitude lower than at one hop.

Ramanathan and Hain [17] report on their ad hoc testbed built with wireless routers. Their aim is to improve QoS in dense networks to be able to use, e.g., voice applications. The authors provide measurements of throughput, end-to-end delay, etc. measured using multiple pings in a few static topologies.

Kaba and Raichle [9], Sanghani et al. [18], Judd and Steenkiste [8], and Ramachadran et al. [16] all describe testbeds where the physical layer effects are emulated in different ways. The aim in all these studies is to mimic mobility in a constrained location for protocol testing purposes.

Zhang and Li [22] have also built an environment for testing large scale ad hoc networks by emulating mobility. Scenario driven packet filtering on source MAC address is used to mimic topology changes. As the authors point out the environment provides merely an approximation of the physical and MAC layers not suitable for performance evaluation.

3. METHODOLOGY AND EXPERIMENTAL SETUP

*University of California Santa Barbara

Experimental studies of wireless ad hoc networks are affected by stochastic factors. The radio environment is hard, or impossible, to control. Another stochastic element is the node mobility. Both these factors have an impact on the network connectivity and thus on the experiment outcome. We need to reduce the variance such that results from different test runs are comparable. The test platform, its software modules, the operating system, and configuration parameters may also interact in an unpredictable way affecting the measurements. Many previous experimental studies [12] [5] [3] do not deal with stochastic factors and how to create repeatable experiments.

We have developed a testbed and an experimental methodology [13] in order to achieve repeatability. In short, our testbed consists of laptops with 802.11 cards, running ad hoc routing protocols. Mobility is created through staff people moving around with the laptops according to instructions on the screens. All test nodes have a homogeneous execution environment realized using our APE testbed [2] as well as the same hardware. The testbed software is a self-contained Linux system including everything needed for the experiments.

In the current study we use identical AST Ascentia P Series laptops (P133MHz) and Lucent IEEE 802.11b WiFi cards (silver). The APE version used is based on version 0.5 and Linux kernel 2.4.26. For the network cards we have the Agere Linux WLAN driver version 7.14 running in IBSS mode with 11 Mbit/s fixed rate, no encryption, and no RTS/CTS. Logging is performed using the *Tcpdump* [19] tool and by the traffic generator. Measurement data is uploaded from each node after the test run, for post-test analysis. We use *Iperf* [6] to generate the TCP traffic and emulate a file transfer. The packet size is set to 1500 bytes (including headers) and packets are transmitted with maximum achievable rate between the source and destination. The Maximum Transmission Unit (MTU) is the default 1500 bytes.

Node mobility can be hard to control in experiments with human test participants. APE allows for specification of detailed choreographic movement instructions that are displayed during the test run. This approach limits the node mobility variance. We reduce external disturbances by running experiments during evenings and nights, avoiding interference from non-test-participants. By promiscuously monitoring data traffic we can study network topology changes in detail and ensure that the protocols in different test runs are exposed to similar connectivity conditions. The scenario descriptions provided below thus illustrate the radio connectivity patterns as opposed to links set up by each routing protocol. We verify that each test run plays out according to these specifications by parsing of the APE log files. If applications crash or terminate prematurely this can be seen in the specific application log. *Tcpdump* log files reveal such cases as well. From the APE log files we generate graphs showing connectivity information which is then used to identify and remove outliers. As an example, when a multi-hop configuration is specified by the scenario, nodes must not overhear routing traffic at a node two hops away, such that it installs a shorter route. We have to accept a certain degree of link fluctuations due to the stochastic radio medium. Currently, we have no automated classification and decision tool to perform the assessment. Instead, this is done manually by inspecting graphs for connectivity and paths taken by the data packets.

4. ROUTING PROTOCOLS STUDIED

In our experiments, we use the following three routing protocol implementations: AODV-UU 0.8.1 [1], UniK-OLSR 0.4.5 [14], and LUNAR (May 2004) [10]. AODV-UU and UniK-OLSR are RFC compliant and interoperability tested. All three protocols are

mature but yet being continuously developed.

Common to the protocols studied is that they use hop-by-hop routing. This means that nodes are unaware of the complete path to the destination. Each routing table entry indicates only the next hop to be used for reaching the final node. We briefly describe the individual characteristics of each protocol in the sections below.

4.1 The Ad Hoc On-Demand Distance-Vector Protocol (AODV)

AODV [15] is a reactive routing protocol. Routes to other nodes are discovered only when needed and maintained for only as long as they are used. For route discovery, the protocol relies on a process with a broadcast Route REQuest (RREQ) and unicast Route REPlies (RREP). Each node traversed by the broadcast RREQ adds a reverse route entry to the source node, with the node it received the RREQ from as next hop. The RREQ is re-broadcast through the network until the sought destination is found, or the Time To Live (TTL) counter expires. If the destination is found, a unicast RREP travels back to the source node, following the reverse path. Each node traversed by a RREP installs a forward route entry to the destination. When a source node receives a RREP it installs the route, which is then ready to be used for sending data.

AODV implementations typically monitor link connectivity by means of periodic broadcast beacons (HELLO packets). Link-layer feedback is an optimization, but is not readily available in most radio modem drivers and therefore not used. A route times out when it is no longer used or if a node does not hear any HELLO packets from the next hop neighbor. Typical values used are a 1 second HELLO interval and a 3 seconds route timeout.

4.2 Optimized Link State Routing Protocol (OLSR)

OLSR [7] is a proactive routing protocol. This means that it maintains routes to all nodes in the network at all times, regardless of if the routes are being used or not. OLSR optimizes the pure link-state algorithm by using *multipoint relays (MPRs)*. Each node selects MPRs among its one-hop neighbors with symmetric, i.e., bi-directional, connectivity. The MPRs are chosen such that they can relay control packets to all two-hop neighbors of the selector node¹.

OLSR performs neighbor sensing by means of broadcast HELLO packets. These are sent periodically by each node (typically every two seconds) and contain lists of neighboring nodes. As with AODV, link layer feedback can instead be used if available.

Changes in link connectivity are carried inside Topology Control (TC) packets. Each MPR periodically broadcasts a TC packet with the link state of its selector nodes. This is typically done every five seconds, or as soon as an MPR detects a change in the MPR set. Other MPRs then propagate these TC packets to all nodes in the network. Nodes use the information in the TC packets to calculate shortest path routes to all nodes in the network.

4.3 Lightweight Underlay Network Ad hoc Routing Protocol (LUNAR)

LUNAR [21] is a hybrid routing protocol, i.e., it operates both reactively and proactively. It is reactive in the sense that it discovers routes only on demand. Similar to AODV, the route discovery is based on broadcast RREQ:s and unicast RREP:s. LUNAR is proactive in the sense that each source node periodically, and unconditionally, re-discovers the route for all route entries that are still

¹A selector node is a regular node which has selected and uses an MPR.

used. The typical periodicity of route re-discovery is three seconds. This periodic re-discovery of active routes means that LUNAR does not have to perform neighbor sensing or route maintenance.

5. DESCRIPTION OF SCENARIOS

The experiments are carried out in an indoor environment using four nodes. It proved to be quite a challenge to achieve comparable tests even with this few nodes why we opted for not studying more advanced scenarios here. However, already at this network size we see that anomalies manifest themselves. If this happens with four nodes we can expect the protocols to malfunction even more as we scale up the experiments.

The physical layout consists of four designated positions which form a multi-hop string topology. Nodes move between the designated positions using varying mobility patterns. The connectivity thus switches between one, two, and three hop configurations. The motivation for this set-up is that we can isolate and study how the protocols perform when links in the forwarding path are lost or when a shorter path becomes available. The scenarios have further been selected to reflect minimal realistic user situations containing periods of multi-hop transmission. In all three scenarios described below there is a single traffic flow between node 3 (source) and node 0 (destination), see Figures 1- 3. We measure the performance of this flow.

5.1 Roaming Node Scenario

The Roaming node scenario consists of three static nodes (labeled 0, 1, and 2) placed in a line at positions A, B and C, respectively. Node 3 moves alongside the static nodes (see Figure 1) with an approximate speed of 1 m/s. The timing is as follows. The mobile node starts moving from position A at time 0 and arrives at position D at time 54. It immediately returns to position A and arrives there at time 108. This ends the test.

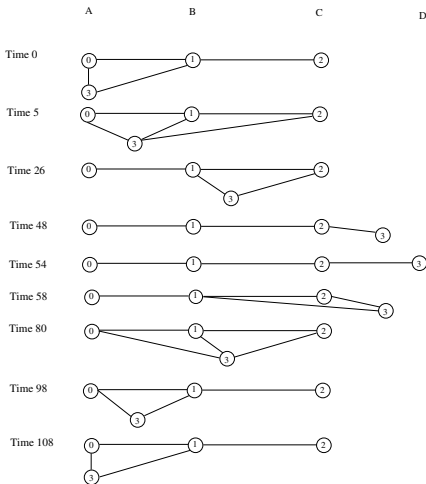


Figure 1: The timing of the basic set of network topology changes that are expected to occur during the Roaming node scenario.

5.2 End Node Swap Scenario

In the start configuration of the End node swap scenario four nodes are placed at positions A, B, C and D (nodes 0-3, respectively). The node mobility consists of the two communicating end nodes (0 and 3) swapping positions once during the test run (see Figure 2). This mobility is a variation of the Roaming node scenario, but here we have two mobile nodes which lose and regain links. This results in different network topology changes and a higher number of link changes compared to the Roaming node scenario. The timing is as follows. The start configuration is held until time 20, at which time the two end nodes start moving towards each other. At time 47 they pass one another and at time 74 arrive at the other end position. This configuration is held for 20 seconds. At time 94 the test ends.

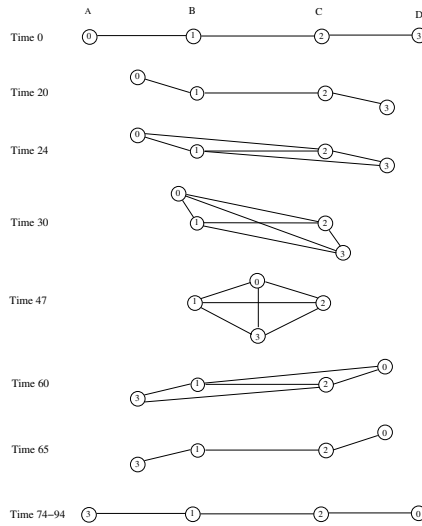


Figure 2: The timing of the basic set of network topology changes that are expected to occur during the End node swap scenario.

5.3 Relay Swap Scenario

The Relay swap scenario is a variant where the communicating nodes are stationary, but the node mobility and connectivity changes occur among nodes along the path. The start configuration and the communicating nodes are the same as in the End node swap scenario. In this scenario, it is the two relay nodes (1 and 2), initially at B and C, that swap positions once during the test run (see Figure 3). The timing is as follows. The start configuration is held until time 20, at which time the two relay nodes start moving towards each other. Around time 32.5 they pass one another and at time 45 arrive at the other relay position. This configuration is held for 20 seconds. At time 65 the test ends.

6. RESULTS AND ANALYSIS

In the experiments, our evaluation metrics are the following.

1. TCP total throughput in each scenario. This measure provides the usability for non-interactive applications.

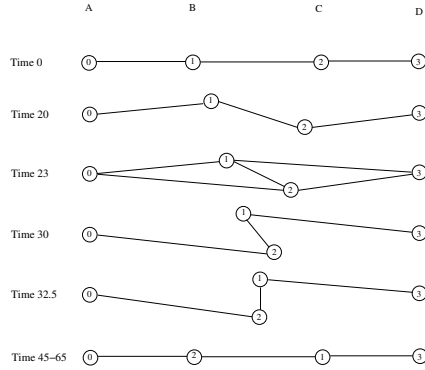


Figure 3: The timing of the basic set of network topology changes that are expected to occur during the Relay swap scenario.

2. TCP no progress times. We thereby take into account sequence number stalls exceeding three seconds. This measure gives an estimation of the user experience in interactive applications that the overall throughput does not capture.

Our results are presented as time versus sequence number at the destination and the accumulated stall time of TCP, i.e., when TCP does not make progress. We expect to see a somewhat stalled TCP due to loss of active paths.

6.1 Roaming Node with TCP

What we ideally expect from the Roaming node scenario is two link breakages when the mobile node moves away and two link reestablishments as it moves back, resulting in the availability of a shortest hop count path going from one to three hops and back.

Figure 4 shows averaged test runs for the Roaming node scenario with AODV-UU, UniK-OLSR and LUNAR, respectively. The communicating nodes, 3 and 0, have direct contact during times 0-26 and 80-108 (see Figure 1). In Figure 4 we see that all three protocols perform well during these periods after TCP has ramped up. Of more interest is the time interval 26-80. Here the packet loss due to link changes and gray zones causes TCP to back-off and eventually stall.

For AODV-UU we see that there is almost no throughput at all until time 90. AODV-UU performs worst of the protocols both in terms of TCP stall time and total amount of data transmitted. In Figure 1 we can see that node 3 regains connectivity with node 1 around time 58. With one-way traffic between node 3 and node 0 this would not make any difference, since AODV-UU on node 3 will keep its route via node 2 until it regains direct contact with node 0. However, with TCP the AODV-UU instance on node 1 will have a routing entry towards node 3. Once node 1 receives a broadcast HELLO packet from node 3 it installs a direct route to node 3. However, at this point node 3 is located in a gray zone with respect to node 1 and can therefore not receive any unicast data packets (TCP ACKs) from node 1.

During the time period 26-80, both UniK-OLSR and LUNAR manages to make some TCP progress. From Figure 4 it looks like UniK-OLSR makes slower but more steady progress, while LUNAR makes more intermittent progress. However, the more distinct steps in the LUNAR graph stem from the fact that we had

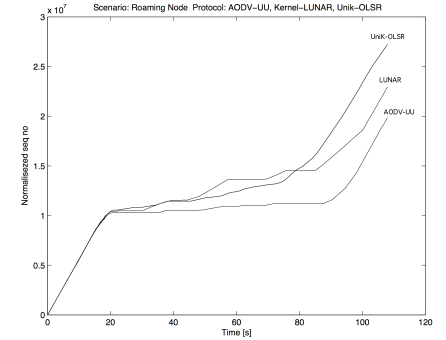


Figure 4: TCP time-sequence graph (mean) for AODV-UU, UniK-OLSR and LUNAR in the Roaming node scenario.

fewer successful test runs with LUNAR. Examining individual test runs reveal that UniK-OLSR makes slightly better progress than LUNAR during this time period. AODV-UU only had one test run which could compare to the performance of LUNAR. We have observed that UniK-OLSR often installs and successfully uses a two hop route during time period 75-85. The different slope of UniK-OLSR's curve during this time period in Figure 4 confirms this.

Table 1: Accumulated TCP "stall times" for the Roaming node scenario.

Protocol	Mean	Median	Std. dev.
AODV-UU	62.1 s	62.1 s	1.8
UniK-OLSR	42.2 s	42.4 s	1.0
LUNAR	49.5 s	49.5 s	2.1

Table 1 shows accumulated no progress results for the Roaming node scenario. We see that AODV-UU performs the worst when using this measure. Its average total TCP stall time is 62 seconds out of 108 which corresponds to 57% of the test run duration. UniK-OLSR and LUNAR perform better, but still stall for 42 and 49 seconds, respectively. This corresponds to 39% and 45% out of the total test run duration.

6.2 End Node Swap with TCP

What we ideally expect from the End node swap scenario is two periods of *multiple* link establishments when the mobile nodes move towards each other (at time 24 and time 30) and two periods of *multiple* link breakages after they have passed and move away (at time 60 and time 65). This results in a quick path change going from three hops, through a short period of two hops to a longer period of one hop. Then the quick path change is repeated in the reverse order. The routing protocols have to react quickly to make use of the short periods of two hop connectivity.

Figure 5 shows averaged test runs for the End node swap scenario with AODV-UU, UniK-OLSR and LUNAR, respectively. We see from this graph that there are two main periods where TCP stalls and makes no progress. The first period is approximately during time 20-35 and the second period is roughly during time 55-75. Looking at Figure 2 we see that the first period corresponds to when the network topology changes from line configuration towards fully connected. The second period corresponds to when the

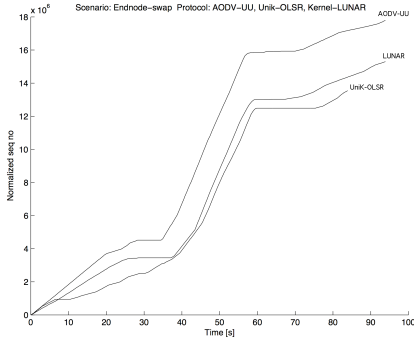


Figure 5: TCP time-sequence graph (mean) for AODV-UU, UniK-OLSR and LUNAR in the End node swap scenario.

network topology changes back to a line configuration.

From Figure 5 we observe that UniK-OLSR has lower performance in the initial phase (time 5-40). We found that UniK-OLSR sometimes removes routes although the proactive link state exchanges indicate connectivity. In several of the test runs it took in the order of seconds before the route re-appeared. We suspect that this is due to an implementation bug, but we have not been able to confirm this. Contrary to what the UniK-OLSR graph in Figure 5 indicates, UniK-OLSR experiences similar TCP stall during the period around time 20-35. However, due to the route loss prior to time 20 the timings of the TCP stalls differs between the test runs. This falsely makes the average graph indicate continuous progress. LUNAR has a single test run which had low, but steady, performance during the initial 25 seconds. This is the reason why the average graph of LUNAR does not match AODV-UU during the initial 25 seconds.

Table 2: Accumulated TCP “stall times” for the End node swap scenario.

Protocol	Mean	Median	Std. dev.
AODV-UU	38.6 s	35.8 s	15.4
UniK-OLSR	43.3 s	41.4 s	7.3
LUNAR	35.2 s	34.6 s	16.5

In Table 2 we see the accumulated no progress results for the End node swap scenario. AODV-UU has an average of 39 seconds out of 94, which translates to 41% of the total test run duration. LUNAR is slightly better with 35 seconds (37%), while UniK-OLSR has 43 seconds (46%). It is noteworthy that although AODV-UU has slightly higher average TCP stall time than LUNAR, AODV-UU outperforms LUNAR with respect to average total throughput. However, from our log files we compared the “best case” test runs and found AODV-UU and LUNAR to be similar.

6.3 Relay Swap with TCP

What we ideally expect from the Relay swap scenario is two link establishments between the mobile relays and their respective opposite end node. Shortly thereafter the relay nodes lose connectivity with their respective original neighboring end node. The path goes from three hops to a short period of two hops and then back. The routing protocols need to react to changes at the intermediate

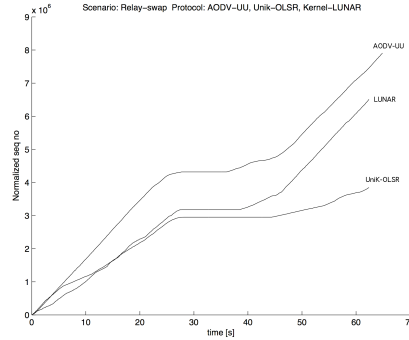


Figure 6: TCP time-sequence graph (mean) for AODV-UU, UniK-OLSR and LUNAR in the Relay swap scenario.

nodes of the multi-hop path.

Figure 6 shows averaged test runs for the Relay swap scenario. We see from this graph that the period of no progress is during time 25-35 for AODV-UU and LUNAR, and during time 25-45 for UniK-OLSR. Performance of both UniK-OLSR and LUNAR varied considerably between the test runs. For UniK-OLSR we often experienced route loss in the order of several seconds for the stationary 3-hop configuration (at time 45-65). This seems similar to the route loss problem we experienced in the initial phase of End node swap. It is notable both in the total throughput and the accumulated TCP stall time.

Table 3: Accumulated TCP “stall times” for the Relay swap scenario.

Protocol	Mean	Median	Std. dev.
AODV-UU	15.7 s	14.1 s	5.8
UniK-OLSR	41.5 s	41.9 s	3.5
LUNAR	14.4 s	14.4 s	4.6

Table 3 shows the accumulated TCP no progress results for the Relay swap scenario. UniK-OLSR has an average of over 41.5(!) seconds out of 65. This corresponds to 65% of the total test run time. We observe that both AODV-UU and LUNAR are notably better in this scenario, with only 15.7 (24%) and 14.4 (22%) seconds stall time, respectively.

6.4 Discussion

It is clear that TCP has problems adjusting to the link changes, even for these simple scenarios and relatively little mobility. Applications thus suffer from very long times without progress. We did not expect TCP to be so significantly affected by the link variations. The reasons for this are complex and require further analysis. One important factor is the gray zones which cause longer times of intermittent connectivity. Such a gray zone can be several seconds long. The routing protocol believes it has a route during this time but no packets get through and TCP gets into back-off mode when there are no ACKs. Once a node has traversed the gray zone, it does *not immediately* start sending packets again. First, it has to wait for the TCP retransmission timer. Since TCP uses exponential back-off, a node can have to wait for a substantial time before starting to send TCP packets again once it is outside the gray zone,

even if a possible path is instantly available.

In our test logs, we observe that OLSR performs route changes later than the other two protocols. This is caused by the longer route time out and lower frequency of HELLO packets of OLSR in comparison to AODV. As mentioned above, LUNAR does not use active neighborhood sensing, but instead rebuilds routes from scratch every three seconds. OLSR further requires links to be symmetric, which takes an extra round of HELLO packets to deduce. Consequently, OLSR often suffers from higher packet loss. In our scenarios, however, OLSR benefits from its route change delay in situations where links are regained such that the route may be optimized. The reason is that OLSR keeps its longer route a couple of seconds more than AODV and LUNAR, and thus OLSR nodes may traverse the gray zone before the protocol changes to the shorter route. Thus, the delayed reaction is in general beneficial for UniK-OLSR when new links are established, but causes severe packet loss when links break.

7. CONCLUSIONS

We have experimentally evaluated the performance of three ad hoc routing protocols. For this purpose we designed three scenarios with simple mobility and traffic patterns. We report on results showing that even using only one mobile node and a single TCP traffic source we see severe performance degradation. We expected TCP to experience problems in regions of connectivity changes due to the restructuring of routes. However, the prolonged stall times of up to about fifteen seconds after connectivity has been regained are surprising.

For most combinations of scenario and routing protocol we experienced TCP stalls for 40%, or more out of the total test run duration. The poor performance can be contributed to a combination of TCPs congestion control and the routing protocols' reactivity and behavior at link changes and in communication gray zones. These elements interact in such a way that TCP stalls for extensive periods although there exist complete and functional path(s) between source and destination. Radio effects such as multipath fading cause varying packet loss during movement, affecting higher layer protocols. At link changes, MAC layer contention is increased because more nodes are in range of each other. Since we are not using RTS/CTS, hidden node problems are also bound to occur in connection with node transitions as nodes move into sensing range of each other causing packet loss to become even more severe. Our conclusion is that the combination of available ad hoc routing protocol implementations, TCP and current MAC layer technology is quite problematic even in small networks with limited node mobility.

The following strategies can be deduced from our measurements. The ad hoc routing protocols tend to optimize to shorter routes before these new routes are stable. This indicates that a hysteresis before installing new links would be beneficial. At the same time, the routing protocols need to be faster to detect and react to link breakages. Our judgment is that basing neighbor sensing solely on HELLO beacons is not sufficiently efficient. The key problems are that they (1) are too slow (several rounds of HELLO messages need to be exchanged which typically takes in the order of seconds), and (2) they are based on broadcast connectivity which might result in gray zone problems. A link-layer feedback method that indicates when transmissions at the MAC layer fail would detect link breakages much earlier. Since the link-layer feedback is based on unicast transmission² and removes the need for HELLO beacons it

²Broadcast transmissions do not require ACKs and the MAC layer and can therefore not deduce whether broadcast packets are suc-

cessfully received. Adding signal strength monitoring would make it more efficient since it then becomes possible to examine whether links with low signal quality can be avoided. We have included link-layer feedback and signal strength monitoring in our AODV-UU implementation and will report on measurements with these functionalities in a forthcoming study.

We are currently in the process of finishing a complementary round of tests to further analyze the reasons for the poor TCP performance. Tests with simpler forms of traffic (UDP and Ping) are included as well as other basic tests aiming at explaining effects caused by the radio medium and MAC layer. All these analyses will be included in an upcoming technical report.

Acknowledgments

The authors wish to thank Fredrik Östergren, Kristoffer Kobosko, and Fredrik Bjurefors for their participation in this experimental study.

8. REFERENCES

- [1] The AODV-UU implementation. <http://core.it.uu.se/adhoc/>.
- [2] APE testbed project webpage. <http://apetestbed.sourceforge.net>.
- [3] B. A. Chambers. The grid roofnet: a rooftop ad hoc wireless network. Master's thesis, MIT, June 2002.
- [4] S. Desilva and Samir R. Das. Experimental evaluation of a wireless ad hoc network. In *Proceedings of the 9th International Conference on Computer Communications and Networks (IC3N)*, October 2000.
- [5] R. Gray, D. Kotz, C. Newport, N. Dubrovsky, A. Fiske, J. Liu, C. Masone, and S. McGrath. Outdoor experimental comparison of four ad hoc routing algorithms. In *Proceedings of the Seventh ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM 2004)*, Venice, Italy, October 2004.
- [6] Iperf webpage. <http://dast.nlanr.net/Projects/Iperf/>.
- [7] P. Jacquet and T. Clausen. Optimized link state routing protocol (OLSR). IETF RFC 3626, October 2003.
- [8] Glenn Judd and Peter Steenkiste. Using emulation to understand and improve wireless networks and applications. In *2nd Symposium on Networked Systems Design and Implementation (NSDI 2005)*, *Usenix and ACM*, May 2005.
- [9] J. Kaba and D. Raichle. Testbed on a desktop: Strategies and techniques to support multi-hop manet routing protocol development. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, October 2001.
- [10] Lunar webpage. <http://core.it.uu.se/adhoc/>.
- [11] H. Lundgren, E. Nordström, and C. Tschudin. Coping with communication gray zones in IEEE 802.11b based ad hoc networks. In *Proceedings of The Fifth ACM International Workshop On Wireless Mobile Multimedia (WoWMoM)*, September 2002.
- [12] D. Maltz, J. Broch, and D. Johnson. Quantitative lessons from a full-scale multi-hop ad hoc network testbed. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, September 2000.

cessfully received.

- [13] E. Nordström, Per Gunningberg, and Henrik Lundgren. A testbed and methodology for experimental evaluation of wireless mobile ad hoc networks. In *Proceedings of First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMMunities (Tridentcom)*, February 2005.
- [14] UniK OLSR implementation. <http://olsr.org>.
- [15] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. IETF RFC 3561, July 2003.
- [16] K. Ramachadran, S. Kaul, S. Mathur, M. Gruteser, and I. Seskar. Towards large-scale mobile network emulation through spatial switching on a wireless grid. In *Proceedings of ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis E-WIND*, August 2005.
- [17] R. Ramanathan and R. Hain. An ad hoc wireless testbed for scalable, adaptive QoS support. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, September 2000.
- [18] S. Sanghani, T. X. Brown, S. Bhandare, and S. Doshi. EWANT: The emulated wireless ad hoc network testbed. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 2003.
- [19] Lawrence Berkeley National Laboratory (LBNL), tcpdump tool download.
<ftp://ftp.ee.lbl.gov/tcpdump.tar.Z>.
- [20] C-K. Toh, M. Delwar, and D. Allen. Evaluating the communication performance of an ad hoc wireless network. *IEEE Transaction on Wireless Communications*, 1(3), July 2002.
- [21] C. Tschudin, R. Gold, O. Rensfelt, and O. Wibling. LUNAR - a Lightweight Underlay Network Ad-hoc Routing protocol and implementation. In *Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN)*, February 2004.
- [22] Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, June 2002.

Paper D

Oskar Wibling, Joachim Parrow, and Arnold Pears. Ad Hoc Routing Protocol Verification Through Broadcast Abstraction. To appear in *Proceedings of the 25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, Taipei, Taiwan, October 2005.

© 2005 Springer-Verlag. Reprinted with permission.

Ad Hoc Routing Protocol Verification Through Broadcast Abstraction

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. We present an improved method for analyzing route establishment in ad hoc routing protocols. An efficient abstraction for Propagating Localized Broadcast with Dampening (PLBD) is developed. Applying this result we are able to verify networks and topology changes for ad hoc networks up to the limits currently envisaged for operational mobile ad hoc networks (MANETS). Results are reported for route discovery in the Lightweight Underlay Network Ad hoc Routing protocol (LUNAR) using UPPAAL and we provide an outline of how similar verifications can be conducted for DSR.

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

1 Introduction

Delivering data in an ad hoc network with mobile nodes requires new protocols. Traditional routing protocols are incapable of routing data packets efficiently in this type of situation, motivating emergence of new protocol proposals. Validation of these new protocols is principally through simulation. Simulation often fails to discover subtle design errors, and therefore formal verification is a promising approach.

In this work, we verify correct operation of the LUNAR [1] protocol route establishment in realistic general scenarios using a network diameter of up to eleven hops. We further describe how the route discovery phase in the DSR [2] protocol can be verified in a similar way. We have aimed for the modeling to be fairly straightforward and for the verification procedure to require a minimum amount of user interaction. The verification properties are formulated at a high and easily assimilated level, e.g. “route possible to set up”.

The operation responsible for most of the complexity in the verification of a LUNAR network scenario is Propagating Localized Broadcast with Dampening (PLBD). PLBD is used in the route discovery phase where a node tries to find a path to another node in the network. Each PLBD phase that is initiated at a node contains a globally unique identifier in order for nodes to keep track of which PLBD:s they have seen. As the name implies, the broadcast propagates through the network, which causes many message exchanges between nodes.

This in turn yields many possible interleavings and leads to exponential growth in verification complexity with regard to increasing number of nodes as well as topological changes in the network.

We show that in any network topology where nodes are positioned so that at a certain time it is *possible* to transmit a message over a link chain between two nodes, the PLBD reaches the intended receiver. Furthermore, we show that if there is at least one such path available then there is always a fastest path. This means that a PLBD initiated by a sender will reach the receiver first along this path. Moreover, all the nodes along the path are the first to receive the PLBD. In LUNAR, network nodes only react to the first specific PLBD they receive; subsequent ones are dropped. Therefore, we model reactions on the first PLBD packet of each type and can safely ignore the rest.

Using this technique, we can model LUNAR with timed automata and perform verifications for realistic network sizes using the UPPAAL [3] tool. Variations on the PLBD operation are also used for route discovery in other ad hoc routing protocols; one example is the DSR protocol. The DSR variant of PLBD differs in that other nodes than the intended receiver can respond to a route discovery, if they happen to possess a cached route to the destination. Therefore, instead of studying just the fastest path, we need to study a number of disjoint paths. This increases the verification complexity, but the saving is still substantial in comparison to studying all possible packet interleavings.

The remainder of this paper is organized as follows. Section 2 covers preliminaries needed to assess the subsequent sections. Section 3 describes our new verification strategy in general and Section 4 provides more detail regarding the actual modeling of LUNAR. Verification results are presented in Section 5. Section 6 describes how the DSR protocol can be verified in a similar way, and Section 7 gives an overview of related efforts in verifying MANET protocols. Finally, Section 8 contains conclusions and describes opportunities for future work in the area.

2 Preliminaries

2.1 Previous Work

In previous work [4] our result was to formally verify important properties of ad hoc routing protocols acting in realistic network scenarios. The scenarios we used are repeated in Figure 1 for clarity. We studied the LUNAR protocol since it combines simplicity with the key properties of more complex ad hoc routing protocols. The protocol was modeled by seeing each node in the network as a separate entity. Each propagating broadcast then hugely increased the complexity because of the possible interleavings of messages. When using this approach we quickly ran out of memory due to verification state space explosion. We were unable to verify networks with more than six participating nodes and very simple topology changes. Here, we refine our method and extend it to verifying significantly larger networks.

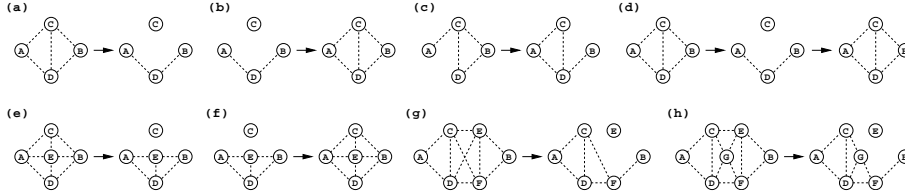


Fig. 1. Example classes of topology changes used in previous work

2.2 The Propagating Localized Broadcast with Dampening (PLBD) Operation

The reason for the state space explosion in the verification of LUNAR is the propagating localized broadcast operation (aka flooding) which works as follows:

- The broadcast is referred to as “localized” since each broadcast in a wireless network only reaches direct neighbors of the transmitting node and not nodes outside transmission range.
- Each node that initiates such a broadcast, tags the broadcast packet with a unique identifier (called the “series selector” in LUNAR).
- At each receiving node the broadcast identifier is compared with a local list to see if this particular packet has been seen before. If that is the case, the packet is just ignored. This mechanism prevents broadcast loops arising in the network.
- If the packet has not been seen before, and the receiving node is not the intended destination, the identifier is stored after which the packet is re-broadcast. Each neighbor, who has not seen the packet before will receive it together with neighbors that may already have seen the packet.
- The intended destination node also stores the identifier when and if it receives the packet, in order to be able to discard subsequent copies it might receive.

In our verification, we assume that this mechanism works, i.e. PLBD can be used as a primitive operation. In making this choice we run the risk of failing to detect subtle operational errors in the PLBD operation, potentially causing failure of the routing protocol. However, this risk is minimized by the analysis of the PLBD process needed to formulate the model.

When using PLBD, the only possible paths that packets can follow from a source to a destination are disjoint. That is, if the destination node receives a number of copies of the same PLBD, these must all have been transmitted through completely disjoint transmission chains. If two transmission chains should coincide at a node, then, since we assume that only one packet at a time can be delivered to each node, one of the packets would have been dropped and the other propagated. Different variants of PLBD have been proposed and used in other protocols [5]. The goal of these is to minimize the number of rebroadcasts needed to reach all connected nodes.

2.3 Brief Description of Ad Hoc Routing and LUNAR

A mobile ad hoc network (MANET) is a transient network that is set up to serve a temporary need, e.g. the exchange of files at a conference. It is assumed that nodes are mobile and that their location can change frequently. Therefore, node connectivity can also vary heavily. In the networks we study, multiple hops are possible between a source and a destination. This, in contrast to a fully connected network, means that nodes outside direct transmission range can be reached by traversing other intermediate nodes. In order to realize this, a routing protocol must be running on each node to forward packets from source to destination.

We study a basic version of the LUNAR protocol and use our earlier pseudo code description [6] to aid us in the modeling. The situation in which we wish to verify correct operation arises when one network node, S , has an IP packet to deliver to another node, D , but there is no route to that node available. In this situation the LUNAR route formation process is initiated at node S , which sends out a route request (RREQ) for the sought node using PLBD. On every retransmitting node, return information for the reply is temporarily stored. If the RREQ reaches D , that node will initiate a unicast route reply (RREP) destined for the node from which it received the RREQ. This node, as well as subsequent ones use the stored return information to re-address the unicast RREP for its next hop. On every step of the way back to node S , relays are also set up for label switching of wrapped IP packets later traveling along the found route. If node S does not receive a RREP within a certain time, it will issue a number of retries (using new PLBD identifiers). After that, the protocol will not take action until there is another IP packet that needs to be delivered.

2.4 General Assumptions

We use the following assumptions throughout this work.

- Unique id:s. It is possible for each network node to generate unique identifiers in isolation from the other nodes. In practice this can be implemented by appending the MAC address of a node to a monotonically increasing sequence number.
- Sequential delivery. Each node in the network can only receive and handle one message at a time. This means that we assume that relatively standard hardware is used in an actual implementation with no parallel processing of messages sent on different channels. Packets thus arrive at each network node in a strict time order.
- Bidirectional links. Only bidirectional links are possible in the network. Since 802.11 requires a bidirectional frame exchange as part of the protocol [2] this is not a significant limitation. It is, however, relevant since it affects the caching strategy of DSR.
- No persistent memory on nodes. If they go down, they lose their current route caches etc.

3 A Refined Modeling Strategy

3.1 Correct Operation of an Ad Hoc Routing Protocol

The definition of correct operation of an ad hoc routing protocol is taken from our previous work [4].

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.

We said that “a path exists between two nodes”, meaning 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.

In the following, we will need a more detailed definition of path existence which pertains only to the unidirectional case. The reason is to be specific about what nodes are connected at different time periods for use in the proofs that follow.

Definition 2. Existence of a unidirectional path

Assume nodes $X_0 = S, X_1, \dots, X_N = D$ (where $N \geq 1$). At time τ_0 a unidirectional path exists from network node S to D if, for all $n \in [0, N - 1]$, between times τ_n and τ_{n+1} node X_n has connectivity and can transmit to node X_{n+1} . Furthermore, between these times, node X_n does not have connectivity to any of the nodes $X_m : m \in [n + 2, N]$.

We require that $(\tau_{n+1} - \tau_n) = T_n : n \in [0, N - 1]$ where T_n is the time required by node X_n to transmit a message to any (or all) of its neighboring nodes (i.e. over one hop), plus the time for the receiving node(s) to handle the packet and prepare for possible retransmission.

Note that we do not limit ourselves to unicast transmission. In the case of LUNAR, the first phase of the route formation process is to send a route discovery along a path from source to destination. We only require this path to be unidirectional. For LUNAR, our previous definition of path existence thus implies Definition 2. Therefore, if the preconditions of Definition 1 hold, then we know there is a unidirectional path at that point in time. In our verification, we will (as before) also make sure that these conditions hold.

3.2 Focusing on the Packet Transformation

We here describe a remedy for the state space explosion problem whilst still being able to model check scenarios of interesting proportion.

In LUNAR, two types of message transfer are used: unicast and PLBD. This is the case for other (reactive) ad hoc routing protocols as well, although some in

addition use regular broadcast e.g. for neighbor sensing. Here, instead of being node centered, we focus on the packet. The idea is that every full (route setup - initial IP packet delivery) session begins with the source node, S , sending out a PLBD packet containing a RREQ for a particular destination, D . When this packet hits one or more other nodes, it can be viewed as being transformed into new packets. Once one of the rebroadcast packets reaches D (provided there is connectivity), this node will generate a RREP unicast packet destined for the node from which it received the RREQ. The RREP then traverses back through the network to S , all the time rewriting addresses. When the last RREP reaches S , it can send its initial IP packet along the found path.

The transformation is probably most easily seen in the case of the unicast chains: one packet comes into a node and another leaves. In the case of broadcast, we would like to be able to ignore all receiving nodes except one, which can then act as a message transformer.

3.3 Disregarding Unimportant Broadcast Receivers

To be able to motivate our claim that we can, at each step, disregard packets received by all broadcast receivers but one, whilst still being able to show important properties of the protocol we will need Theorem 2 below. First, however, we need a theorem that guarantees that we find a path if we use PLBD.

Theorem 1. Existence of a PLBD path

In a finite mobile network, if there at time τ_0 exists a possible unidirectional path between two nodes S and D , according to Definition 2, then a PLBD initiated from node S at this time will reach node D . The PLBD path is then the inverse of the following sequence of nodes: Node D , the node that broadcast the PLBD to D , and so on all the way back to node S .

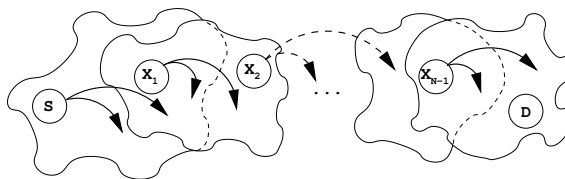


Fig. 2. Localized broadcast traversing network

Proof. See Figure 2 for an illustration. A PLBD with unique identifier β initiated by node S at time τ_0 will reach the direct neighbors of S . According to Definition 2 these direct neighbors either contain node D , or some intermediate node X_1 . If node D was reached directly, it cannot have seen the PLBD with identifier β before and will receive the packet.

If node D was not in the direct neighbor set of S , then we know (according to our definition) that one of the other nodes in this set, say X_1 , will receive the PLBD and, at time τ_1 be able to retransmit it to its neighbors. These neighbors may partially overlap the neighbor set of node S , but, according to Definition 2 the set will either contain node D or at least one other node, X_2 , that has not previously seen this PLBD (with id β).

Continuing in this fashion, since the network is finite, we will eventually have transmitted to the final connected node(s) that had not yet heard the PLBD with identifier β . According to our definition, node D will then also be among the nodes that have received this PLBD.

Definition 3. Fastest path

A path ξ between two nodes S and D is faster than another path ρ at some point in time, if, at this time it is possible to deliver a packet from S to D faster along path ξ than along path ρ . A path χ is the fastest path between two nodes at some point in time, if, at this time it is faster than all other paths between the two nodes.

Theorem 2. Uniqueness of a PLBD path

If there at one point in time exists at least one PLBD path, from one network node to another one, then during the same time there must exist exactly one PLBD path.

Proof. Because the number of loop free paths in a finite graph is finite, the number of paths between two nodes, S and D , in a finite network is also finite. Then, if packets are sent from node S along all (disjoint) paths, one of them will be the first to reach node D , namely the one sent along the fastest path. The fastest path will also be the unique path, since node D will henceforth disregard all PLBD packets it receives containing that particular identifier.

Thus, to recapitulate, PLBD:s can effectively be studied as propagating unicasts. We illustrate this for LUNAR in Figure 3, describing the protocol with the help of a message sequence chart. We can see that, for the case of LUNAR, it is completely packet driven and in essence only reacts to incoming packets by updating internal tables and generating a new unicast or PLBD packet. If we always study the fastest path for every PLBD we cannot get any interference from other copies of the same broadcast packet since these will be dropped everywhere but along the fastest path according to our assumptions. Nodes that are not on the fastest path will therefore not be part of a chain forwarding the packet all the way to the intended destination node. Thus, we have fully motivated our packet transformation model and can go on to describing the model itself. What we essentially do is to reduce it from a parallel to a sequential one whereby complexity is significantly reduced.

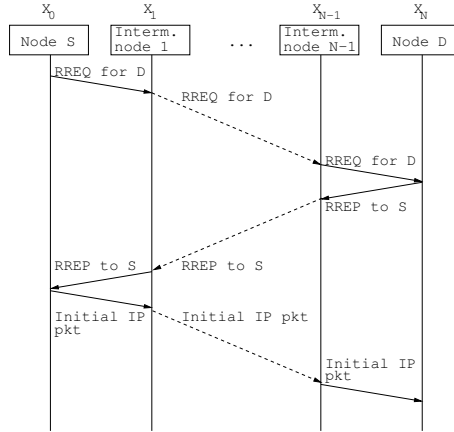


Fig. 3. LUNAR MSC with RREQ PLBD only along fastest path

4 Modeling Approach

4.1 The UPPAAL LUNAR Model and Verification Properties

UPPAAL [3] is a tool for simulation and verification of systems modeled as timed automata. Verification properties are passed to the system as Linear Temporal Logic (LTL) formulae. We have chosen to use UPPAAL in our work because of its powerful model checking capabilities and because we can use time in our models in a straightforward way. This further enables us to extract time bounds for route formation and initial IP packet delivery.

The LUNAR timed automata model includes a template (`lunar_message`) which models a packet in transit between two nodes. We use a system definition with three processes, representing the initial route discovery and two retries. Thus, a delay is passed as a parameter in order to model the timeout triggered resend. As in previous work we do not model any expanding ring search, but use a timeout value of 75 ms corresponding to three times the ring timeout in current LUNAR implementations and settle for two resends. Time only passes when messages are transmitted between network nodes, and we have used a range of [2,4] ms to model this delay. This represents four to eight times the theoretical delay lower limit (DLL) for the basic access mechanism of IEEE 802.11b [7]. Note that intermittent transmission failures on lower layers (e.g. due to packet collisions) are treated as link breakages in our model. In addition to the general assumptions in Section 2.4 we assume route caches to be initially empty.

Our model is to some extent less abstract than in previous work since we now model the selector tables explicitly. This is done through arrays (since there are no more complex data structures available), but it is still feasible since we gain state space usage from the PLBD abstraction. When a packet arrives at a node it needs to be switched so as to use new selectors. These are modeled using

global arrays that for each node (MAC address) map selector value to a (MAC address, selector) pair.

Along the path of a PLBD, symbolic addresses of the intermediate nodes are generated as we go. These can be seen as pointers to the real addresses. Therefore, we select them from a limited range of numbers, e.g. [0,8] if we admit a maximum of 9 intermediate nodes along the fastest path. For each new route request the symbolic addresses are selected from different ranges, even though they may in reality point to the same node. Errors due to subtle faults in the algorithm that allocates selectors might elude our analysis as a result of this assumption.

We choose to verify deadlock freedom as well as route formation and initial IP packet delivery. These are verified by checking that we can eventually get to the `snd_node_rec_lunar_rrep` (sender node received LUNAR RREP) and `message_del` (IP packet delivered) states along all execution paths. To extract the time bounds, a global timeout is used and experimentally tuned for the upper range. For the lower range we instead use LTL formulae to check possibility for route formation and initial IP packet delivery along at least one execution path.

4.2 Correspondence between Scenarios

Instead of specifying each individual scenario exactly, we are now able to parameterize on the following:

- Maximum network diameter (number of hops), d_{max} . The maximum number of possible intermediate nodes on the unique PLBD path between source and destination, plus one.
- Number of possible link failures, f , during playout of the scenario. Note that these represent critical link failures in the sense that we model them by dropping a packet nondeterministically along the fastest path.
- (Minimum network diameter, $d_{min} \leq d_{max}$; but this value should be set to 1 to allow for all possibilities of communicating nodes' positions. The only time we use a different value is when checking correspondence to previous scenarios where positions of source and destination nodes were specified.)

The scenarios we can study using the new model encompass all the ones in our previous work (shown in Figure 1). Our definition does not require the routing protocol to find a route (or send an initial IP packet along the route) if all paths are broken. We can include link breaks if we make sure that the protocol is given the chance to find a path along some other route, in accordance with the requirement of Definition 2. The inclusion of link breaks is important in order to verify that the protocol copes with those situations as well, in the case of LUNAR by initiating another route discovery after a timeout.

Scenario (g) in previous work corresponds to setting up our new model with minimum and maximum path lengths of three and with one possible link break. This is because in scenario (g) there is one link break that can occur at any time. The minimum and maximum path lengths are three, both before and after the

link break. As an illustration see Figures 4 and 5 which show two possibilities for the packet traversal. Here, solid lines denote packets that are delivered, and dotted lines denote packets that are dropped because the receiver has already seen that particular PLBD identifier. Other traversals are possible and node E may go down at other times, but because of the dampening, it should be quite clear that the maximum path length will be three regardless of the order in which packets are delivered.

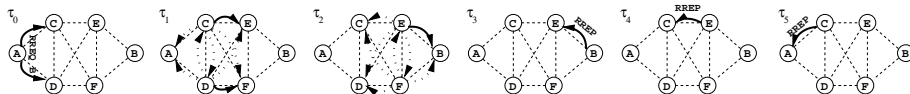


Fig. 4. Stepwise traversal of scenario (g) - Route setup initiated before link break

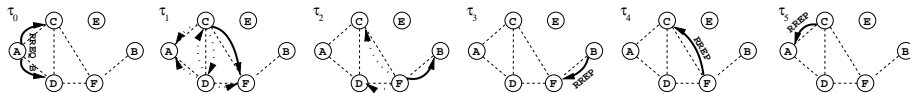


Fig. 5. Stepwise traversal of scenario (g) - Route setup initiated after link break

We validate that this is correct with our new model by extracting a bound on initial message delivery time, which is [18,111] just as in our previous work. Using the same reasoning, we can easily translate all the previous scenarios to parameters in our new model.

5 Verification Results and Analysis

We have performed verification of LUNAR networks for the properties of interest (see Section 4.1). For general networks, i.e. where $d_{min} = 1$, we are able to verify route setup up to a diameter (d_{max}) of eleven hops, when using $f = 1$. For the same value of f we can verify initial IP packet delivery using $d_{max} = 8$ before running out of memory. This greatly surpasses the network size for which LUNAR is meant to operate (3 hops), this limit being due to the so called ad hoc network horizon [1]. Each verification takes less than a few minutes on a Macintosh PowerBook G4 laptop computer with a 1.33 GHz processor and 1.25 GB of memory. We also include some measurements from using the same processing power and verification software configuration as in our previous work. These data are presented in Table 1 together with our previous results to illustrate how substantial the performance increase is.

Due to space constraints, we are here only able to include one of our result plots. Figure 6 shows bounds on route formation and initial IP packet delivery times for the case when $d_{min} = d_{max}$ and $f = 1$. The same results as in corresponding scenarios of our previous work are obtained. As mentioned, we can also

Table 1. Comparison of UPPAAL verification results

Scenario	Explicit broadcast modeling			Using broadcast abstraction		
	States searched	Time used	Search completed	States searched	Time used	Search completed
(a)	15072	3.89 s	Yes	487	< 1 s	Yes
(e)	123196	57.91 s	Yes	487	< 1 s	Yes
(g)	2.01e+06	11:43 min:s	Yes	910	< 1 s	Yes
(h)	2.97e+07	1:59 h:min	No	910	< 1 s	Yes

verify the more general case where $d_{min} = 1$, and the difference in time bounds is that they then reach down to 4 ms for the route setup time and to 6 ms for the initial message delivery. The reason is that the most extreme case then is when source and destination are in direct contact, whereby the route setup can be completed in two transmissions. With this setting of d_{min} the verification also includes all intermediate situations, which increases the complexity.

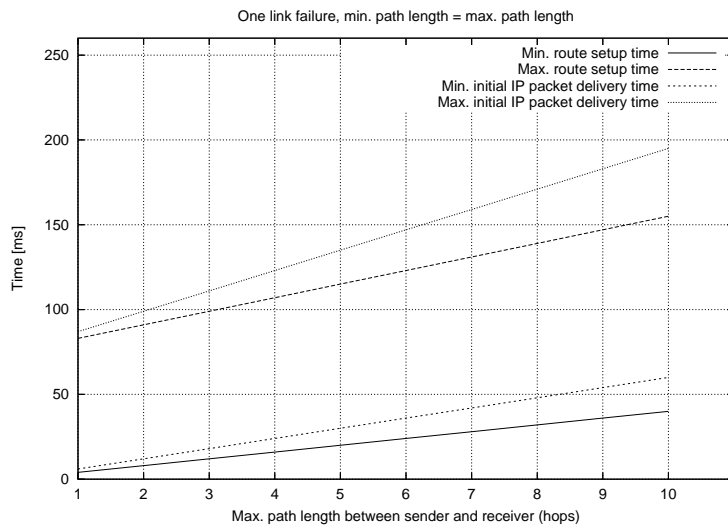


Fig. 6. Example plot showing time ranges extracted

Thus, since we are able to use a rather high number for the network diameter we are in fact able to study all networks of practical significance even though we have not produced a general proof. Our results are general in the sense that any mobility model can be accommodated. We only require that it, at some point, yields a network configuration with a unidirectional path between a given pair of nodes. Link breakages affect the initial state of route caches in the network.

It is therefore important to study if different settings for f cause the protocol to behave differently. When increasing f from 1 to 2, and using $d_{min} = 1$, we can verify route setup up to $d_{max} = 8$ and initial IP packet delivery up to $d_{max} = 6$. Introducing more link breaks thus reduces the maximum network diameter that can be used. Due to the routing protocol structure of LUNAR its worst case behavior is captured by admitting the same number of link breaks as resends. The protocol cannot be expected to guarantee successful route discovery if more link breaks occur, since all retries may then be lost. Given that we only model two LUNAR resends (see Section 4.1) we therefore choose to set the limit at $f \leq 2$ in this study.

We can perform verification of all the scenarios studied in previous work, but considerably faster. Furthermore, we are able to study more general network configurations of much greater size. In a related project we are currently performing real world experimental evaluation of a number of ad hoc routing protocols. There, scenarios with a maximum of four nodes are used and even then, we note trouble in forming multi-hop paths which causes severe performance penalties for TCP [8].

6 Comparison of Route Discovery in DSR and LUNAR

DSR [2] and LUNAR can both be considered as on-demand protocols because neither of them relies on any kind of periodic packets to be exchanged between nodes. In the basic route discovery phase, the two protocols operate in a similar way. However, there are some important differences:

- DSR is a source route protocol which means that the RREQ packet includes addresses of all the nodes passed along the path from source to destination. This list is then returned to the source node and used as header for each IP packet that is subsequently to be routed. At each step, nodes use the next address in the header as new destination. In the case of LUNAR, label switching is instead used in nodes for the rewriting of addresses.
- LUNAR only stores the first response received from a route discovery. In DSR, on the other hand, a node may learn and cache multiple routes to any destination. This is also possible through overhearing routing information from packets sent by others as opposed to in LUNAR where nodes only use information they have themselves requested.
- In DSR, nodes which are not themselves the sought destination may answer with one of their routes. The answer will contain the list of addresses traversed thus far concatenated with the cached route. Loop segments are identified and removed, and a node cannot return a route in which it is not itself included.

In a DSR model we need to account for these differences properly. Instead of one destination for the PLBD, we need to study a set of answering nodes, \mathcal{D} . As an upper bound for this set we have the number of disjoint paths originating from network node S . We can, however, settle for all those that reach a neighbor

of D , since the others will not be valid at the time of the RREQ. In the DSR draft it is said that the number of hops will often be small (e.g. perhaps 5 or 10 hops). It is also stated that the DSR protocol is designed mainly for mobile ad hoc networks of up to about 200 nodes. In a finite network, the set of answering nodes is also finite. The maximum value will be the total number of nodes in the network minus one (the sending node). This case appears if S and D are directly connected and all other nodes are also connected to both the source and destination. The implication for verification is, however, not as severe as it may first seem since no path can then be longer than two hops.

7 Related Work

Chiyangwa and Kwiatkowska [9] have studied timing properties of AODV [10] using UPPAAL. Their model uses a linear topology with specialized sender, receiver and intermediate nodes. The authors investigate how network diameter affects the protocol. They report that at 12 intermediate nodes, the recommended setting for route lifetime starts to prevent long routes from being discovered. They propose adaptive selection of this parameter to compensate for the behavior in large networks. This work is related to ours, but the linear scenario type contains a static number of nodes and its motivation is to discover a maximum diameter. Apart from providing a formal motivation to a single network path, our methodology encompasses a variety of topologies. Their method involves constructing a specialized model where we use the same protocol instance at each node which simplifies the modeling process.

Obradovic et al [11] have used the SPIN [12] model checker and the HOL [13] theorem prover to verify route validity and freedom from routing loops in AODV. They used conditions on next node pointers, sequence numbers and hop counters to form a path invariant on pairs of nodes (on the path from source to destination). Three lemmas were then verified using SPIN after which HOL was used to prove that the three lemmas imply the path invariant theorem (using standard deductive reasoning). The approach requires a significant amount of user interaction and is not directly applicable to other protocols.

Das and Dill [14] also prove absence of routing loops in a simplified version of AODV. The strategy is similar to that of Obradovic et al, but more automated. They use predicate abstraction and can discover most of the quantified predicates automatically by analyzing spurious abstract counter-example traces, albeit with some mechanical human involvement. 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. Proficiency in formal verification is required in order to make general use of their method.

de Renesse and Aghvami [15] have used SPIN to model check the ad hoc routing protocol WARP. They use a general 5-node topology, and provide a non-exhaustive verification (using the approximating bitstate hashing mode [12]), covering 98% of the state space.

Xiong et al [16, 17] have modeled AODV using Petri nets. A topology approximation mechanism describes dynamic topology changes. They report on a looping situation found during a state space search of a general ten node topology. Their broadcast model uses an average number of messages based on the average degree and the total number of nodes in the graph. The resulting PLBD implementation is less abstract than ours and models redundant packet transfers between nodes not on the fastest path between the sender and receiver. In contrast to our approach link failure effects are also not included in their model as they assume unicast transmissions to be globally receivable regardless of topology.

With our method we can use the same protocol instance for each symbolic node and easily verify high level properties of ad hoc routing protocols, such as “initial IP packet delivered”. We do not put strict requirements on the topologies, but admit for general networks of a certain diameter and a given number of link breakages. The modeling and verification processes are thus quite simple and applicable to a range of different protocols.

8 Conclusions and Future Work

We have developed a new efficient method for modeling PLBD, one of the operations in ad hoc routing protocols most responsible for state space explosion in previous approaches. We applied the technique, verifying the operation of route establishment in the ad hoc protocol LUNAR, and derived upper and lower time bounds for both route establishment and first packet delivery over the resulting route. We verified route setup in networks of up to eleven hops in diameter, well over the envisioned upper limit for practical application of ad hoc routing in realistic scenarios.

For our future work we intend to perform the same verification for the DSR protocol, which we have only sketched here. We want to see if there are other protocols that utilize primitive operations, responsible for much of the complexity, which can be abstracted away from in order to enable for verification in realistic networks. These analyses can further be used to compare the quality of competing protocols.

References

1. Tschudin, C., Gold, R., Rensfelt, O., Wibling, O.: LUNAR: a lightweight underlay network ad-hoc routing protocol and implementation. In: Proc. Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN). (2004)
2. Johnson, D.B., Maltz, D.A., Hu, Y.C.: Internet draft: The dynamic source routing protocol for mobile ad hoc networks (DSR). <http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt> (2004)
3. Larsen, K.G., Petterson, P., Yi, W.: UPPAAL in a Nutshell. Int. Journal on Software Tools for Technology Transfer **1** (1997) 134–152

4. Wibling, O., Parrow, J., Pears, A.: Automatized verification of ad hoc routing protocols. In: Proc. 24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE). (2004)
5. Ni, S.Y., Tseng, Y.C., Chen, Y.S., Sheu, J.P.: The broadcast storm problem in a mobile ad hoc network. In: Proc. ACM MobiCom. (1999) 151–162
6. Wibling, O.: LUNAR pseudo code description. http://user.it.uu.se/~oskarw/lunar_pseudo_code/ (2005)
7. Xiao, Y., Rosdahl, J.: Throughput and delay limits of IEEE 802.11. IEEE Communications Letters **6** (2002) 355–357
8. Lundgren, H.: Implementation and Experimental Evaluation of Wireless Ad Hoc Routing Protocols. Phd thesis, Uppsala University (2005)
9. Chiyangwa, S., Kwiatkowska, M.: A timing analysis of AODV. In: Proc. Formal Methods for Open Object-Based Distributed Systems: 7th IFIP WG 6.1 International Conference (FMOODS). (2005)
10. Perkins, C.E., Royer, E.M.: Ad hoc on-demand distance vector routing. In: Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications. (1999) 90–100
11. Obradovic, D.: Formal Analysis of Routing Protocols. Phd thesis, University of Pennsylvania (2002)
12. Holzmann, G.: The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, Reading, Massachusetts (2003)
13. University of Cambridge Computer Laboratory: Automated reasoning group HOL page. <http://www.cl.cam.ac.uk/Research/HVG/HOL/> (2005)
14. Das, S., Dill, D.L.: Counter-example based predicate discovery in predicate abstraction. In: Formal Methods in Computer-Aided Design, Springer-Verlag (2002)
15. de Renesse, R., Aghvami, A.: Formal verification of ad-hoc routing protocols using SPIN model checker. In: 12th Mediterranean Electrotechnical Conference (IEEE MELECON). (2004)
16. Xiong, C., Murata, T., Tsai, J.: Modeling and simulation of routing protocol for mobile ad hoc networks using colored Petri nets. In: Proc. Workshop on Formal Methods Applied to Defence Systems in Formal Methods in Software Engineering and Defence Systems. (2002)
17. Xiong, C., Murata, T., Leigh, J.: An approach to verifying routing protocols in mobile ad hoc networks using Petri nets. In: Proc. IEEE 6th CAS Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication. (2004)

Recent licentiate theses from the Department of Information Technology

- 2003-012** Olivier Amoignon: *Adjoint-Based Aerodynamic Shape Optimization*
- 2003-013** Stina Nylander: *The Ubiquitous Interactor - Mobile Services with Multiple User Interfaces*
- 2003-014** Kajsa Ljungberg: *Numerical Methods for Mapping of Multiple QTL*
- 2003-015** Erik Berg: *Methods for Run Time Analysis of Data Locality*
- 2004-001** Niclas Sandgren: *Parametric Methods for Frequency-Selective MR Spectroscopy*
- 2004-002** Markus Nordén: *Parallel PDE Solvers on cc-NUMA Systems*
- 2004-003** Yngve Selén: *Model Selection*
- 2004-004** Mohammed El Shobaki: *On-Chip Monitoring for Non-Intrusive Hardware/Software Observability*
- 2004-005** Henrik Löf: *Parallelizing the Method of Conjugate Gradients for Shared Memory Architectures*
- 2004-006** Stefan Johansson: *High Order Difference Approximations for the Linearized Euler Equations*
- 2005-001** Jesper Wilhelmsson: *Efficient Memory Management for Message-Passing Concurrency — part I: Single-threaded execution*
- 2005-002** Håkan Zeffer: *Hardware-Software Tradeoffs in Shared-Memory Implementations*
- 2005-003** Magnus Ågren: *High-Level Modelling and Local Search*
- 2005-004** Oskar Wibling: *Ad Hoc Routing Protocol Validation*



UPPSALA
UNIVERSITET