



Mojgan Kamali

Formal Analysis of Network Routing Protocols

TURKU CENTRE *for* COMPUTER SCIENCE

TUCS Dissertations
No 242, August 2019

Formal Analysis of Network Routing Protocols

Mojgan Kamali

*To be presented, with the permission of the Faculty of Science and
Engineering of Åbo Akademi University, for public criticism in Auditorium
XX, the Agora building, on August 23rd, 2019, at 13:00.*

Åbo Akademi University
Faculty of Science and Engineering
Agora building, Vattenborgsvägen 3,
20500 Åbo, Finland

2019

Supervisor

Adjunct Professor Luigia Petre
Faculty of Science and Engineering
Åbo Akademi University
Agora building, Vattenborgsvägen 3, 20500 Turku
Finland

Reviewers

Professor Peter Csaba Ölveczky
Department of Informatics
University of Oslo
Ole-Johan Dahls Hus, Gaustadalleen 23 B, 0373 Oslo
Norway

Associate Professor Cristina Secoleanu
Networked and Embedded Systems (NES) Division
Mälardalen University
School of Innovation, Design, and Engineering, Västerås
Sweden

Opponent

Professor Peter Csaba Ölveczky
Department of Informatics
University of Oslo
Ole-Johan Dahls Hus, Gaustadalleen 23 B, 0373 Oslo
Norway

ISBN 978-952-12-3844-4
ISSN 1239-1883

To Mom, Dad and Jan

Abstract

The use of wireless networks has been on the rise for some time now, from the ubiquitous smart phones and laptops in use everywhere, to sensor networks collecting large amounts of data. In this dissertation, we focus on contemporary wireless technologies, in particular Wireless Mesh Networks (WMNs): self-organising and self-healing wireless networks that support broadband communication without requiring any wired infrastructure. A significant factor for the *reliability* and *flexibility* of such networks is provided by the routing protocols. The current approaches used for analysing routing protocols, e.g., test-bed experiments and simulation techniques, are expensive, time consuming and resource-intensive. Additionally, these techniques cannot, in general, guarantee the reliability and flexibility of such systems. In order to address these challenges, the use of formal methods is growing, i.e., for the purpose of reasoning about wireless mesh network routing protocols. However, the mathematical foundation of formal techniques is considered as a challenging task for protocol designers, and therefore, this level of development (reliability and flexibility analysis) is often skipped.

The objective of this dissertation is to study different routing protocols of wireless mesh networks using formal techniques as well as to propose a generic framework to formally model, analyse and verify such protocols as our ultimate goal. The proposed framework can help the protocol designers to verify their routing protocols prior to implementing them in real-case scenarios and to avoid consequent costs. We employ classical model checking (via the Uppaal tool), statistical model checking (via the Uppaal SMC tool) and theorem proving in Event-B (via the Rodin platform) to carry out our study and deal with both quantitative and qualitative analysis of such systems. Our generic and reusable framework is developed employing the Uppaal SMC as the tool support.

Sammanfattning

Användningen av trådlösa nätverk har ökat kontinuerligt under en längre tid, från vanliga smarttelefoner och bärbara datorer, till sensornätverk som samlar stora mängder data. I denna avhandling fokuserar vi på modern trådlös teknik, i synnerhet trådlösa mesh-nätverk: självorganiserande och självlärande trådlösa nätverk som stöder bredbandskommunikation utan behov av någon kabelbaserad infrastruktur. En betydande faktor för *pålitlighet* och *flexibilitet* av sådana nätverk tillhandahålls av routingprotokollen. De nuvarande metoderna som används för att analysera routingsprotokoll, t.ex. testbäddsexperiment och simuleringstekniker, är dyra, tidskrävande och resurskrävande. Dessutom kan dessa tekniker inte i allmänhet garantera tillförlitligheten och flexibiliteten hos sådana system. För att ta itu med dessa utmaningar ökar användningen av formella metoder, dvs. att matematiskt resonera om trådlösa routingprotokoll. Den matematiska grunden för formella tekniker betraktas emellertid som en utmanande uppgift för protokolldesigners, och därför uteblir denna nivå av utveckling (tillförlitlighet och flexibilitetsanalys) ofta.

Syftet med denna avhandling är att studera olika routingprotokoll för trådlösa mesh-nätverk med hjälp av formella tekniker samt att föreslå ett generiskt ramverk för att formellt modellera, analysera och verifiera sådana protokoll som vårt ultimata mål. Det föreslagna ramverket kan hjälpa protokolldesigners att verifiera sina routingprotokoll innan de implementeras i riktiga användningsscenarier och därmed hjälpa till att undvika följdkostnader. Vi använder klassisk modellkontroll (via Uppaal-verktyget), statistisk modellkontroll (via Uppaal SMC-verktyget) och teorem bevisning med Event-B (via Rodin-plattformen) för att genomföra vår studie och hantera både kvantitativ och kvalitativ analys av sådana system. Vårt generiska och återanvändbara ramverk är utvecklat med hjälp av Uppaal SMC som verktygsstöd.

Acknowledgements

First of all, I would like to express profound thanks to my supervisor Docent Luigia Petre for her excellent advice, support and her continuous encouragement. Furthermore, I wish to thank Prof. Peter Csaba Ölveczky and Associate Prof. Cristina Seceleanu for their valuable reviews of this dissertation and for providing constructive comments that improved its quality. Particular thanks are due to Prof. Peter Csaba Ölveczky for also agreeing to act as an opponent at the public defence of the thesis.

I am grateful to all members of the Faculty of Science and Engineering at Åbo Akademi University, especially my colleagues at the Distributed Systems Laboratory. In particular, I wish to thank my friend Dr. Johan Ersfolk for his help with practical matters and the Swedish version of the abstract. I would like to acknowledge Prof. Massimo Merro's kind help and supervision during my stay at Verona University. I extend my sincere thanks to Prof. Ansgar Fehnker for his supervision and friendship during my stay at Twente University. My appreciation also goes to Prof. Joost-Pieter Katoen for his support and supervision during my stay in RWTH Aachen University. I am very grateful to all my co-authors for the knowledge they shared with me.

I gratefully thank the Faculty of Science and Engineering at Åbo Akademi University for the generous funding of my doctoral studies and travels. I would like to acknowledge the Centre for International Mobility (CIMO), the Nokia foundation and the Finnish Foundation for Technology Promotion (TES) for granting me research scholarships that supported my work. I also wish to thank Prof. Olaf Owe for his financial support during my stay at the Department of Informatics, Oslo University.

Finally, I would like to express my thankfulness to my family and my friends for their cheeriness and support throughout these years. I wish to thank my best friend, Shima, for her continuous encouragement and heartening. I am especially grateful to my brothers, Ehsan, Morteza and Paul, and my dear sister, Maryam, who continuously motivated me to keep the pace in my research. I would like to express my deepest gratitude to my dearest parents, Nahid and Ali, for their love and supporting me spiritually throughout my life. I will be grateful forever for your support. Last but not least, I wish to thank my beloved best friend, Jan, who has been constant source of inspiration and support.

Mojgan Kamali
Åbo, August 2019

List of original publications

- I Mojgan Kamali, Peter Höfner, Maryam Kamali and Luigia Petre, Formal Analysis of Proactive, Distributed Routing, In Radu Calinescu and Bernhard Rumpe (Eds.) *Proceedings of the 13th International Conference on Software Engineering and Formal Methods (SEFM 2015)*, Lecture Notes in Computer Science Vol. 9276, pp. 175-189, Springer, 2015.
- II Mojgan Kamali and Luigia Petre, Improved Recovery for Proactive, Distributed Routing, *Proceedings of the 20th International Conference on Engineering of Complex Computer Systems (ICECCS 2015)*, IEEE proceeding, pp. 206 - 209, IEEE, 2015.
- III Mojgan Kamali and Luigia Petre, Modelling Link State Routing in Event-B, *Proceedings of the 21st International Conference on Engineering of Complex Computer Systems (ICECCS 2016)*, IEEE proceeding, pp. 207-210, IEEE, 2016.
- IV Mojgan Kamali and Luigia Petre, Uppaal vs Event-B for Modelling Optimised Link State Routing, In Kamel Barkaoui et al. (Eds.) *Proceedings of the 11th International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS 2017)*, Lecture Notes in Computer Science Vol. 10466, pp. 189-203, Springer, 2017.
- V Mojgan Kamali, Massimo Merro and Alice Dal Corso, AODVv2: Performance vs. Loop Freedom, In A Min Tjoa, et al. (Eds.) *Proceedings of the 44th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2018)*, Lecture Notes in Computer Science Vol. 10706, pp. 337-350, Springer, 2018.
- VI Mojgan Kamali and Ansgar Fehnker, Adaptive Formal Framework for WMN Routing Protocols, In Kyungmin Bae and Peter Ölveczky (Eds.) *Proceedings of the 15th International Conference on Formal Aspects of Component Software (FACS 2018)*, Lecture Notes in Computer Science Vol. 11222, pp. 175-195, Springer, 2018.

Contents

Part I: Research Summary	1
1 Introduction	3
2 Wireless Mesh Networks (WMNs)	7
2.1 Routing Protocols	7
2.1.1 Proactive Routing Protocols	8
2.1.2 Reactive Routing Protocols	10
3 Background: Formal Methods	15
3.1 The Uppaal Model Checker	16
3.1.1 Scalability with Classical Model Checking	18
3.2 Uppaal Statistical Model Checker	19
3.3 Event-B	20
4 Contributions	23
4.1 Paper I: Formal Analysis of Proactive, Distributed Routing	23
4.2 Paper II: Improved Recovery for Proactive, Distributed Routing	24
4.3 Paper III: Modelling Link State Routing in Event-B	25
4.4 Paper IV: Uppaal vs Event-B for Modelling Optimised Link State Routing	26
4.5 Paper V: AODVv2: Performance vs. Loop Freedom	27
4.6 Paper VI: Adaptive Formal Framework for WMN Routing Protocols	28
5 Related Approaches	31
5.1 Model Checking	31
5.2 Statistical Model Checking	33
5.3 Theorem Proving	34
6 Discussion	37
6.1 Summary	37
6.2 The Challenge	37

6.3	What we have achieved	38
6.4	Strengths and Limitations	39
6.5	Future Directions	40
7	Bibliography	41
	Complete List of Publications	49
	Part II: Original Publications	51

Part I

Research Summary

1. Introduction

Wireless technologies have been on the rise for some time now: laptops and smart phones are ubiquitous, wireless sensor networks monitor the environment and generate vast quantities of data, and envisioned electrical cars wirelessly recharge [63]. Due to the wireless aspect, these technologies are always subject to failure under adverse conditions. To demonstrate *resilience*, i.e., the maintenance of network services to an acceptable level even when failures occur, *reliability* and *flexibility* are needed. In our work, we focus on contemporary wireless technologies, in particular *wireless mesh networks* (WMNs). We study their reliability by analysing their performance w.r.t. data delivery assurance as well as their flexibility by analysing their dynamic adaptability to changes.

Wireless mesh networks are self-organising and self-healing networks that bear the benefit of low-cost and rapid deployment. These networks have gained popularity and are increasingly applied in a wide range of application areas, including:

- *Public safety*: governments are responsible for providing public safety in order to protect their citizens. For instance, *closed circuit televisions* (CCTVs) are used in this regard.
- *Military communication*: WMN devices attached to soldiers and devices allow for their coordination and monitoring in the battlefield.
- *Environmental monitoring*: WMNs allow observing local conditions, for instance in case of natural disasters (fire, flood, etc) or hardly accessible places (volcanoes).
- *Railway and transportation*: WMNs provide the possibility to monitor railways and to detect malfunctioning of such systems.

WMNs deployment is envisioned to continue in the future. Reliability and flexibility evaluations for WMNs will play crucial roles in the design process for such systems, and will ensure their successful development in practice. This includes cases where networks inevitably become more complex and where node mobility can significantly affect the performance of a

network. Data communication may fail due to such dynamic behaviour of the network and, as a consequence, catastrophic results can occur. Therefore, it is clear that a priori evaluations are of high importance for safety critical applications of WMNs.

Researchers believe that shortcomings of contemporary WMN systems are due, to a large extent, to limitations of the current network protocols and their inability to tailor and adapt their operation to the very different and dynamic deployment environments of WMNs [64]. Therefore, investigating the behaviour of wireless network protocols is important in order to understand and overcome the limitations of current networks. For example, a *routing protocol* enables node communication in a network by disseminating information that enables nodes to select routes. In this way, nodes are able to send data packets to arbitrary destinations in the network. This makes routing protocols one of the key factors determining the performance of WMNs. These protocols are classified into two main categories: *proactive* and *reactive*. Proactive protocols rely on the periodic broadcasting of control messages through the network and have the information available for routing data packets. Reactive protocols, in contrast, behave on-demand, meaning that when a packet targeting some destination is injected into the network they start the route discovery process.

Typically, the analysis of network protocols consists of test-bed experiments and simulation techniques. These are appropriate techniques for performance analysis and typically employ synthetic models of the network and its dynamic characteristics, for system validation. However, when considering reliability and flexibility, these approaches are limited in the following ways:

- The underlying mathematical models are often vague and unavailable to users. Such models are often unrealistic, and the results obtained can vary between different simulators. As a consequence, the results depend on the underlying model of the simulator, rather than on the characteristics of the routing protocol under study.
- Simulation tools do not sufficiently support computational behaviours, which again leads to unrealistic results.
- The optimisation of a routing protocol requires a thorough comparison of different designs, for example when system performance depends on the choice of system parameters. With simulation, such comparative analysis requires the statistical interpretation of a large number of simulation runs, making the analysis difficult and costly.

The question is how to develop these protocols in a reliable and flexible way so that we can trust the developed system to behave as expected by

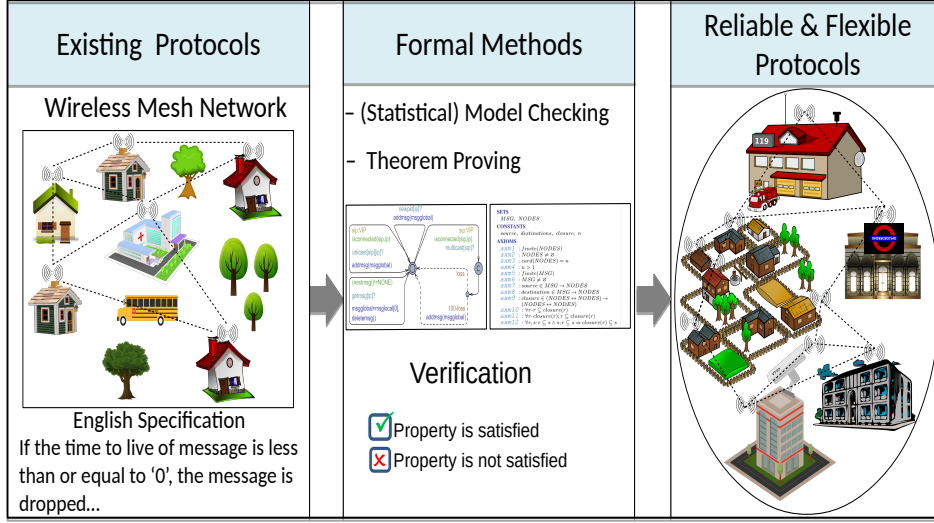


Figure 1.1: Our overall goal.

the specification. This question applies to the entire lifecycle of the protocols, from specification to deployment, and includes the investigation of alternatives to improve system qualities. The overall goal of our research is to contribute to the development of reliable and flexible WMN routing protocols. The impact of this diverse area of research can be enormous because of the ubiquity of such systems in our society.

Our research approach is based on applying *formal techniques* in order to achieve our research goals. We use formal techniques on the existing routing protocols of WMN in order to improve their reliability and flexibility (Figure 1.1). Formal techniques have usually precise mathematical syntax and semantics, and allow the analysis of WMN routing protocols in a more rigorous manner when compared to traditional simulation and test-bed approaches. *Formal methods* can be considered as a special class of formal techniques that provide valuable tools for designing, developing, comparing and evaluating systems [51]. They can provide critical assurance w.r.t. performance of the system under study. Formal methods have been successfully used in many application areas [44, 53, 50, 49], and have proved particularly useful in the very early design stages, when only a model or a blueprint of the product is available.

Concretely, we formally model and verify different routing protocols of WMNs by applying different formal techniques. We investigate how different classes of such routing protocols (proactive and reactive) behave with respect to different system requirements (quantitative and qualitative analysis). We have used different formal modelling and analysis frameworks (a classical as well as a statistical model checker and a theorem prover) in order to find

out which formalism is the most suitable when having such systems under study. A complementary aspect of our work consists of developing a formal generic framework to model, analyse and verify WMN routing protocols. In this manner, protocol designers can adapt the generic models based on the protocol specifications and verify routing protocols prior to implementation.

Our research is summarised as a collection of six papers. We divide this dissertation in two parts. We describe the overall view of our work in Part I, and Part II follows as a reprint of research papers (with permission). In Part I we proceed as follows. In Section 2, we overview WMNs and their routing protocols and in Section 3 we describe the research methods that we used. In Section 4, we sum up and explain how we employed the methods from Section 3 to the WMN routing protocols described in Section 2; paper by paper, we characterise our contribution to formal modelling and verification of WMN routing protocols. In Section 5, we discuss related work relevant to our contributions and in Section 6 we summarise our work as well as sketch future research directions.

2. Wireless Mesh Networks (WMNs)

In this section we explain the main characteristics of wireless mesh networks and their routing protocols, to the extent needed this thesis.

A WMN consists of a set of nodes arranged in a mesh topology: richly interconnected to each other [57] via uni-directional or bidirectional wireless links. These nodes communicate with other nodes in their transmission range via messages. The WMN *topology* refers to the physical layout of the nodes and their connectivity, modelled as a graph $G = (N, L)$ with the finite set of nodes N and the finite set of links L .

A mesh network topology tends to remain static (except for the random failure of links or addition of new links) in order to cover route computation as well as to deliver data packets to specified destinations. A mesh network allows redundancy, meaning that when a node is no longer operating, the other network nodes are still able to communicate with each other. This is either directly or through intermediate nodes, demonstrating the self-forming and self-healing characteristics.

Nodes send control messages in order to obtain information about the network. A *message* is a communication unit contributing to the functionality of a network. It contains a source node, a destination node, and possibly some data. A *routing algorithm* provides routes in the network that can be taken by messages to reach their destinations.

In this dissertation, we aim at modelling, analysing and verifying selected WMN routing protocols as well as at developing a generic and reusable framework for the development of such protocols. We deal with both proactive and reactive classes of routing protocols, described in the next section.

2.1 Routing Protocols

Routing protocols specify routes taken by messages sent between nodes. The routes are usually discovered by distributing node information via messages through the network. Neighbouring nodes receive these messages, enabling them to obtain information about possible routes. In this way, different routes between nodes are discovered, established and possibly maintained. Each node in the network keeps a *routing table* to store information

on routes to other nodes. The information in the routing table entries is generally modelled as a tuple with at least the following components: $(dip, hops, nip, dsn)$ where:

- dip stands for the address to an arbitrary destination node (say d)
- $hops$ shows the distance (number of hops necessary) to reach d
- nip is the next neighbour node in the route towards d
- dsn represents the last sequence number received from d by the node the routing table is for, describing how recent the information is: the higher the sequence number, the more recent the route.

The information in the routing tables of the nodes describes at best a partial configuration with respect to the connectivity in the network, as it was at some point in the past; in the most general scenario, node and link failures continually change that configuration.

Routing protocols are grouped into two main categories, namely *proactive* and *reactive*. Proactive protocols work in a foreseeing way, starting route discovery in advance, before any data packet is received. Reactive protocols behave on demand, meaning that they start discovering routes upon receiving a data packet destined for some destinations. In the following, we describe each category of routing protocols and the two main protocols that have been the focus of this dissertation.

2.1.1 Proactive Routing Protocols

Proactive protocols keep lists of destinations together with routes to them by flooding control messages through the network. Nodes broadcast control messages periodically, to build and update their topological information about the other nodes of the network so that, when a data packet is injected into the network it can be rapidly routed to the destination node. Some often used proactive protocols are: *optimised link state routing* (OLSR) [19] and *better approach to mobile ad-hoc networking* (BATMAN) [48]. We describe the behaviour of the OLSR protocol with an example in Figure 2.1. The picture shows a network containing 4 nodes connected to each other in a linear topology running the OLSR algorithm, with a data packet from s to d injected at a random time. Representation of the OLSR control messages is also shown in the Figure. Field * depicts recipients of (re)broadcast control messages that are nodes in the transmission range of the sender node.

The proactive nature of OLSR implies the benefit of having the routes available when needed. The underlying mechanism of this protocol consists in the periodic exchange of two types of messages, namely *hello* and *TC (topology control)* messages relying on the so-called *multi-point relays*

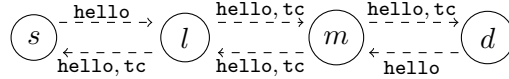
$$\begin{array}{lll}
s \longrightarrow * & : & \text{hello}, s, \text{Onehops} \\
l \longrightarrow * & : & \text{hello}, l, \text{Onehops} \\
m \longrightarrow * & : & \text{hello}, m, \text{Onehops} \\
d \longrightarrow * & : & \text{hello}, d, \text{Onehops} \\
l \longrightarrow * & : & \text{tc}, l, \text{Sseq}, \text{hc}, \text{sender}, \text{mprs}, \text{ttl} \\
m \longrightarrow * & : & \text{tc}, m, \text{Sseq}, \text{hc}, \text{sender}, \text{mprs}, \text{ttl}
\end{array}$$


Figure 2.1: The OLSR routing protocol.

(*MPRs*) in order to find routes. For every node there are one-hop neighbours that are MPR nodes, meaning that an MPR node has bi-directional links towards some of the two-hop neighbours of that node. Based on the OLSR specification, only MPRs (re)broadcast TC messages through the network, optimising the traffic in the network.

The message flow is represented by dashed arrows in Figure 2.1, denoting the broadcast of *hello* messages: $\langle \text{hello}, \text{originator}, \text{Onehops} \rangle$, and *topology control* (TC) messages: $\langle \text{tc}, \text{originator}, \text{Sseq}, \text{hc}, \text{sender}, \text{mprs}, \text{ttl} \rangle$. The field **hello** shows the *type* of a message, field **originator** models the message *generator*, and field **Onehops** contains the list of the one-hop neighbours of the message generator. The TC message has the message *type* **tc** and is generated by the node stored in field **originator**, field **Sseq** models the *originator sequence number* to be added in the routing table entries pointing towards the TC originator, field **hc** shows the number of *hops* from the TC originator to the receiving node, the **sender** field presents the TC *sender* which is rebroadcasting the TC, field **mprs** contains the list of nodes that have selected the TC generator as their MPR, and field **ttl** is time to live, representing the maximum number of hops a message can be transmitted.

Every node broadcasts a *hello* message every 2 seconds in the network and detects its direct neighbours by receiving these messages. As a *hello* contains information about one-hop neighbours of the originator, it allows the receiving nodes to know about their two-hop neighbours. A *hello* traverses only one wireless link or a single hop and it is not forwarded. This type of message is used for neighbour detection and for selecting MPR nodes.

We assume node *s* broadcasts a *hello* of the form $\langle \text{hello}, s, \emptyset \rangle$. The **Onehops** set is empty at the starting point of the protocol. The other nodes *l*, *m* and *d* also follow the same procedure as of *s*, broadcasting *hello* messages every 2 seconds through the network. When a *hello* is received by the

intermediate node l , it updates its routing table for s , marking s as its one-hop neighbour. Later when l broadcasts its own *hello* through the network, it embeds the information about s as it is the one-hop neighbour of l . The *hello* from l has the form $\langle \text{hello}, l, s \rangle$, where l is the *hello* originator and s is the one-hop neighbour of l .

When s receives the *hello* from l , it updates its table corresponds to l , marking it as its one-hop neighbour, and when m gets the *hello*, it updates its table for both s and l as the information about s was also provided in the received *hello* from l . Then m learns that l is the MPR node because it has a link toward s which is the two-hop neighbour of m . The process repeats for all network nodes and by this, nodes learn about their one-hop neighbours, two-hop neighbours and their MPRs.

Finally, nodes l and m , selected as MPRs, broadcast TC messages every 5 seconds to build and refresh the topological information. The TC message from l has the form $\langle \text{tc}, l, 1, 0, l, \{s, m\}, 3 \rangle$. When the neighbour node m receives the TC from l , it acts as follows:

1. It looks for the pair $(l, 1)$ in its routing table to investigate if it has already processed the TC. If this is the case, the TC message is discarded and processing of the message stops. Otherwise, the pair is subject to enter the routing table. Hence, the routing table is updated for l .
2. Then, m rebroadcasts the TC from l with the modified information as follows: $\langle \text{tc}, l, 1, 1, m, \{s, m\}, 2 \rangle$.

This procedure is also followed by the other MPR nodes in the network (node m) and thus they obtain the network information, so that when a data packet is injected into the network, it can be routed toward its destination.

2.1.2 Reactive Routing Protocols

Reactive protocols look for routes by flooding on-demand route requests through the network. This means that when a data packet aimed for some destinations is injected into the network, the protocol initiates the route discovery process to deliver the data packet to its destination. There are numerous reactive protocols such as: *ad-hoc on-demand distance vector* (AODV) [55], *dynamic manet on-demand* (DYMO) [56], *dynamic source routing* (DSR) [34]. We describe the behaviour of the DYMO protocol with an example illustrated in Figure 2.2. The picture shows a network containing 4 nodes connected to each other in a linear topology running the DYMO routing algorithm; s is the source node, d is the destination node, l and m are the intermediate nodes. Representation of the DYMO control messages is also shown in the figure. The field $*$ depicts recipients of (re)broadcast control messages that are nodes in the transmission range of the sender node.

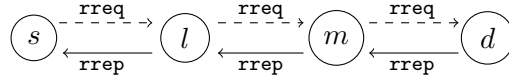
$$\begin{array}{lll}
s \longrightarrow * & : & \text{rreq}, s, d, \text{Sseq}, \text{Dseq}, \text{intms}, \text{hc} \\
l \longrightarrow * & : & \text{rreq}, s, d, \text{Sseq}, \text{Dseq}, \text{intms}, \text{hc} \\
m \longrightarrow * & : & \text{rreq}, s, d, \text{Sseq}, \text{Dseq}, \text{intms}, \text{hc} \\
d \longrightarrow m & : & \text{rrep}, s, d, \text{Sseq}', \text{intms}, \text{hc} \\
m \longrightarrow l & : & \text{rrep}, s, d, \text{Sseq}', \text{intms}, \text{hc} \\
l \longrightarrow s & : & \text{rrep}, s, d, \text{Sseq}', \text{intms}, \text{hc}
\end{array}$$


Figure 2.2: The DYMO routing protocol.

The message flow is represented by dashed and continuous arrows, respectively showing broadcast and unicast communication. The *RREQ* (*route request*) messages defined as $\langle \text{rreq}, \text{originator}, \text{destination}, \text{Sseq}, \text{Dseq}, \text{intms}, \text{hc} \rangle$ are broadcast and the *RREP* (*route reply*) messages represented as $\langle \text{rrep}, \text{originator}, \text{destination}, \text{Sseq}', \text{intms}, \text{hc} \rangle$ are unicast.

The *rreq* and *rrep* fields represent the *type* of the message. Fields *originator* and *destination* show the message *source* and *destination*, respectively. The *Sseq* field denotes the *source sequence number*, i.e. the sequence number to be added in routing table entries pointing towards the originator node. The *Dseq* field contains the *destination sequence number* showing the latest sequence number received by the originator node in the past for any route towards the destination; if destination is unknown to the originator this number is 0.

The field *intms* denotes a set of *intermediate* nodes on the way (accumulated path) from the message originator to the receiving node. DYMO uses the concept of *path accumulation*: whenever a control message travels via more than one node, information about *all* intermediate nodes on the route is stored in the message. In this way, a node receiving a message establishes routes to *all other intermediate nodes*. Initially, there is no entry when the originator broadcasts RREQ. The path is accumulated later when the RREQ is rebroadcast via intermediate nodes. The *hop-count* field *hc* represents the number of hops from the message originator to the node receiving the request. Initially, this field is set to 0.

Assume the source node *s* has a data packet for the destination node *d*. In order to send the data packet, *s* looks for an entry for *d* in its routing table. If such an entry does not exist, it will start the route discovery process to find a route to *d*. The protocol continues as follows:

- The source *s* generating a RREQ increases its own sequence number and broadcasts the RREQ of the form $\langle \text{rreq}, s, d, 1, 0, \emptyset, 0 \rangle$.

- When RREQ is received by the intermediate node l , it acts as follows:
 1. It looks for the pair $(s, 1)$ in its routing table to investigate if it has already processed the request. If this is the case, the RREQ is discarded and processing of the message stops. Otherwise, the pair is subject to enter the routing table and the routing table is updated for s .
 2. Then, l looks for an entry for d in its routing table. If such an entry exists, with destination sequence number greater than or equal to RREQ's destination sequence number, l increases its own sequence number and two RREP messages are sent. One is sent to s saying that l is the next node toward d , and another one is sent to d saying that l has a path to s . If l does not have any entry with a destination sequence number greater than or equal to the **Dseq**, it adds itself as the intermediate node of RREQ and then re-broadcasts the RREQ with the **hc** field incremented by one as $\langle \mathbf{rreq}, s, d, 1, 0, \{l\}, 1 \rangle$.
- Node m later repeats the same procedure as node l .
- Whenever d receives the RREQ, it checks if the message has been already processed. If this is the case, the message is discarded. Otherwise, d updates its routing table for s and intermediate nodes l and m . Then, d increases its sequence number and sends a unicast RREP $\langle \mathbf{rrep}, s, d, 1, \emptyset, 0 \rangle$ to m . Here, s and d are copied from the incoming RREQ and sequence number of d is possibly updated according to d 's sequence number which is unique to d . The *intms* represents intermediate nodes from the message originator toward the receiving node. Initially, there is no entry and the path is accumulated later when the RREP is unicast via intermediate nodes. The *hop-count* field is set to 0.
- The RREP then follows the reverse path towards node s increasing the **hc** field by one passing each hop. Each node receiving the reply packet will update the routing table entry associated with d and *intms* if one of the following conditions is met: (i) route to d is unknown; (ii) the sequence number in the RREP associated with d is greater than the one exists in the routing table; (iii) the sequence numbers are the same but the new route is shorter. In this way, nodes on the reverse route toward s learn the route to d and intermediate nodes *intms* on the way. As soon as the route discovery from s to d is finished, the injected data packet to s can be routed toward its destination d .

Nodes also monitor the status of alternative *active* routes to different destinations. Upon detecting the loss of a link in an active route, a *route error* (RERR) message is broadcast to notify the other nodes about the link failure. The RERR contains the information about those destinations that

are no longer reachable toward the broken link. When a node receives an RERR from its neighbours, it invalidates the corresponding route entry for the unreachable destination.

3. Background: Formal Methods

In this section we overview the formal tools that we apply in this dissertation.

Formal methods are broadly categorised into different categories, e.g., event-based and state-based. Event-based formal methods model systems as a composition of several processes handling events and communicating via channels. Some examples of event-based approaches are CSP [29], π -calculus [47], and CCS [46]. State-based formal methods model the state of the system and specify methods modifying these states. Some examples of state-based methods are Action Systems [5], the B-method [1], the Z notation [61]. There are also some studies on combining event-based and state-based methods proposed as CSP||Event-B [59] and CSP||B [58].

A common feature of all formal methods is the concept of a *formal* (or *precise*) *model* associated with a meaning called *semantics*: mapping a novel concept to a concept which we have already understood well. Based on formal models we can specify precise properties of systems and verify if they hold for a certain model; we can also capture different versions (at different levels of abstraction) of the same system. These formal models can be interpreted similarly by different users.

Two important issues formal methods had to deal with for a long time are those of usability and scalability. As the formal semantics is based on mathematics and logic, specifying systems in a formal manner requires a certain formal background.

Thus, an essential instrument for the usability and scalability of formal methods consists in tool support that usually provides a platform to cope with system modelling and analysis as well as to deal with syntax checking. They can also prove essential properties of the system, fully or partially automatic. For example, model checking techniques prove system properties automatically, whereas proving properties is partially automatic in theorem proving techniques (they sometimes need interaction from the modeller). In this dissertation, we employ the event-based formalism of *timed automata* [4], supported by the Uppaal model checker [10] and by the statistical extension of Uppaal, namely the Uppaal-SMC [22]. We also employ the state-based formal method Event-B [2], supported by the Rodin platform [3].

We apply Uppaal to model the timing behaviour, wireless communica-

tion, and complex data structure of routing protocols. The continuous timing mechanism provided by Uppaal allows us to precisely model the timing aspects of the protocols. Synchronisation mechanisms provided by Uppaal, i.e., broadcast and binary synchronisation, allow us to precisely model the wireless communication between nodes. Common data structures of Uppaal, such as structs and arrays, and a C-like programming language are used to model routing tables and update-operations on such tables. In addition, the Uppaal GUI and Uppaal simulator provide a visualised interpretation of the system which makes the task of modelling easier. Uppaal carries out an exhaustive exploration of the state space of the model in order to guarantee that the system does not violate system requirements (properties).

Uppaal SMC, an extension of Uppaal model checker, is employed in order to model probabilistic behaviour of communication protocols in addition to the features mentioned above, using stochastic timed automata. For instance, message loss which is a common phenomenon in wireless communication can be rigorously modelled using probabilities. Uppaal SMC also overcomes the barrier of analysing large systems and it provides both quantitative and qualitative analysis.

We also apply the Event-B formalism to manage the complexity of routing protocols into distinct abstraction layers. We start with a very abstract model and gradually add more details to the models to make them more concrete, using the refinement technique. Each model is verified prior to adding any additional feature to guarantee the correct by construction design. Scalability issues that Uppaal cannot handle are also dealt with in Event-B, as the size of the model is not an issue.

In the following we overview these formal methods and shortly their associated tools.

3.1 The Uppaal Model Checker

Uppaal [10] is a well-established model checker providing support for modelling real-time systems as networks of (extended) timed automata that can synchronise on channels and shared variables. These models are used for analysis and formal verification. A finite timed automaton [4] is represented as a graph consisting of finite sets of locations (nodes) and edges (transitions), together with a finite set of clocks having real-valued numbers. The logical clocks of automata are initialised to zero and are increased with the same rate. Each location may have an invariant, and each edge may have guards that capture the conditions for traversing edges (possibly clock guards which allow the progress of time), and/or actions that can be updates of some variables and/or resetting clocks.

A *timed automaton* is defined as a tuple (L, l_0, C, A, E, I) where: L is a

finite set of locations, l_0 is the initial location and is an element of L , C is a finite set of clocks, A is a set of actions, co-actions and internal actions, $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with actions, guards and a set of clocks (note that $B(C)$ defines finite sets of guards on edges given as clock constraints), $I : L \rightarrow B(C)$ are invariants on locations. For instance, going from location l_i to l_j is possible if an edge from E can be taken, satisfying its guard from $B(C)$. If so, the action on that edge can happen.

The Uppaal modelling language extends timed automata introduced by Alur and Dill [4] with additional features such as data structures, types, etc. These features provide the ground to model complex behaviours of routing protocols. Global declarations and processes are distinguished in Uppaal. All processes are running concurrently at the same level, and the model has no further hierarchy. Processes are in fact instantiations of parameterised templates. Separate from the system model are the queries describing system properties and requirements.

Type definitions – often used as identifiers – define ranges of integers in the global declaration. Variables can have integer type, newly introduced types, channels and arrays of these. Clock variables are applied to measure time and they evaluate to real numbers. Progress of time happens at the same rate for all clocks and can be reset only to zero. Functions can also be defined in a C-like language as parts of declaration and can be used elsewhere in the model.

Every process contains its own local declaration of variables and functions that are limited to the process. For every process, there exists an automaton operating on global and local variables and functions. Channels are responsible for synchronisation between automata, meaning that for each channel c there exists one label $c!$ identifying the sender and $c?$ presenting the receivers. Transitions having no label are internal transitions and all the other transitions use either broadcast or binary handshake synchronisations.

Binary handshake (unicast) synchronisation means that one automaton that has an outgoing edge with a $!$ -label synchronises with another automaton having a $?$ -label edge, if guards on both edges hold in the current state. When the transition is taken, both automata move to their next locations, and variables are modified according to the updates on edges; first, variables on $!$ -label edge are updated, then $?$ -label edge follows, updating its variables. When the choice is on more than one possible pair, the choice is made non-deterministically.

Broadcast synchronisation means that one automaton that has an outgoing edge with a $!$ -label synchronises with several other automata having edges with a relevant $?$ -label. The automaton with a $!$ -label edge moves

to its next location, and update its variables if and only if its guards are satisfied. It does not require a second automaton to synchronise with. Other matching ?-label edge automata are required to synchronise if their guards hold in the current state. They will move to their next locations and update their variables. First, the variables on the !-label edge are updated, then the other matching automata with ?-label edge follow. If the choice is on more than one initiating automaton (the automaton with an outgoing ?-label edge), the choice is made non-deterministically.

System properties (requirements) can also be defined in addition to the system model and they can access all local and global variables and functions, if they are side-effect-free (if they do not change variables outside of the scope of functions). Verification in Uppaal is carried out employing a decidable fragment of Computation Tree Logic (CTL) to state system properties. CTL provides two types of formulas, namely *state* and *path* formulas. State formulas deal with individual states in the model without considering the behaviour of the model, whereas path formulas quantify over paths, containing an (infinite) sequence of states in the model. A path formula can be of three main types, namely *reachability*, *safety* and *liveness*.

A reachability property checks whether or not there is any reachable state in the system that can satisfy the given state formula ϕ . This property is expressed as $E <> \phi$ using Uppaal syntax, stating that there exists (E) at least one path that eventually ($<>$) satisfies ϕ . A safety property essentially states that something bad never happens. In Uppaal, this property is formulated positively saying that something good invariantly happens, either for all paths or for at least one path. If property ϕ is true for all reachable states (\Box) for all paths (A), it is stated as $A\Box\phi$. If the property ϕ is true for all reachable states (\Box) in at least one path (E), it is stated as $E\Box\phi$. A liveness property states that something will eventually ($<>$) happen in all paths (A). Given state formula ϕ , the liveness property called “leads to” is expressed as $A <> \phi$ in Uppaal. The other form of liveness property is the form of $\phi \rightarrow \varphi$, meaning that whenever ϕ is satisfied, then φ will be eventually satisfied.

3.1.1 Scalability with Classical Model Checking

Depending on the designed model, the number of states in the model can easily become enormous. As an example, having a system composed by m processes, each containing n number of states, the asynchronous composition of the processes can contain n^m states that are required to be explored exhaustively. This problem is referred as the *state space explosion* [18] and is a common issue in all model checkers. Researchers have developed several techniques in order to address this problem, however the problem still

exists. For example, partial order reduction [54], bounded model checking [14], counterexample-guided abstraction refinement [17], etc combat the state space explosion problem, however they cannot overcome the problem entirely. Therefore, we have used Statistical Model Checking and Event-B as alternatives to cope with this problem.

3.2 Uppaal Statistical Model Checker

Uppaal SMC [22] is considered as a trade-off between simulation and model checking techniques in order to overcome the size barrier of classical model checking and to provide probabilistic reasoning. The main idea of Uppaal SMC is to observe only some simulation traces of the system and to apply Monte Carlo simulation or sequential hypothesis testing, respectively for quantitative and qualitative analysis. This is done in order to decide whether or not the system can satisfy the intended property with a given degree of confidence.

The specification language of Uppaal SMC is similar to Uppaal model checker described earlier in this chapter, i.e., based on extended timed automata. Uppaal SMC assigns probabilities to the different enabled transitions of automata; thus, the non-deterministic choices between these transitions is replaced by choosing the transition with the highest probability and choices of delays are made randomly. A system is modelled as a network of stochastic timed automata [12], communicating with each other via broadcast channels and shared variables. Uppaal SMC has additional query language elements of the Metric Interval Temporal Logic (MITL) in order to provide support for estimating and comparing probabilities, hypothesis testing and also evaluating expected values.

A system modelled as networks of stochastic timed automata is input for verification to provide statistical evidences for either satisfaction or violation of desired properties. Then the probability that a system satisfies the defined property is computed. For example regarding routing protocols, the probability that a route is established from a source node to a destination node, or the probability that an injected data packet can be delivered to its destination can be computed. After conducting some simulations of the system, it can be statistically verified whether or not the simulations satisfy defined properties. Verification does not give us a 100% guarantee w.r.t. properties, however it is possible to limit the interval in which an error can occur.

In order to estimate probabilities, the algorithm computes the number of runs that are required for defining an approximation probability interval $[p - \epsilon, p + \epsilon]$ in which ϕ (state formula) with a confidence of $1 - \alpha$ is satisfied. Here, p stands for the average probability over all runs, the value of ϵ shows

the probabilistic *uncertainty* and α presents the *false negatives*. These values are used for specifying the statistical confidence of the result. The values of α and ϵ are selected by the user and the number of runs is then calculated by the tool applying the Chernoff-Hoeffding bound.

In Uppaal SMC, the corresponding query is stated as $\text{Pr}[bound](\phi)$, where *bound* presents the simulations time bound and ϕ defines the expression (path formula). So, the probability that ϕ is satisfied or violated in *bound* is calculated. Uppaal SMC also provides support for evaluating expected values of a maximum of an expression that can be an integer or a clock. The number of runs N and the *bound* should be explicitly entered in order to evaluate the maximum of the given expression *expr*. In Uppaal SMC, the corresponding query is expressed as $\text{E}[bound; N](\max : expr)$. The values of α and ϵ should be provided explicitly by the user.

3.3 Event-B

Event-B [2] is a formal method based on the B-Method [1] and the Action Systems [5], employed for modelling and analysing distributed systems. Event-B provides automated tool support via the Rodin [3] platform. This allows to specify and verify distributed systems by formally modelling the system and to prove that the constructed model fulfils some specified system properties.

Event-B uses the concepts of *context* and *machine* modules in order to define the specification of the system (systems behaviour) as well as to state properties (requirements) of the system. A *context* in Event-B represents the static part of the system, dealing with carrier sets and constants that are used to state axioms in the model, whereas a *machine* concerns the dynamic part of the system, modelling the system and its properties using *variables*, *invariants* and *events*. A *machine* can access its context by defining the *Sees* keyword.

The state of a system is described by the values of variables that are modified by the system events using *actions*. Each event can have *guards*, which are logical properties that limit the event: it can only be executed if its guards are evaluated to *true*. If there are several enabled events in the system, the choice of execution is made non-deterministically. Invariants of the system define the system properties and they must hold for any reachable state in the system before and after the execution of events.

Event-B uses the refinement [7] concept to gradually add details into an initial abstract specification of the system. An abstract and non-deterministic machine M_0 is refined by a more concrete machine M_1 (defined as $M_0 \sqsubseteq M_1$), having new variables and events, so that the correctness of M_0 is preserved during its transformation to M_1 .

There are different types of refinement, for instance in *superposition* refinement, events in M_0 can be refined in M_1 , enriching their behaviour with new variables behaviour whereas *data refinement* replaces some variables of M_0 by other variables in M_1 and specifies *gluing invariants* to describe the relation between old and new variables. All occurrences related to old variables of events in M_1 are substituted by newly defined variables and only the gluing invariant contains old variables. *Proof obligations* are automatically generated by the Rodin platform in order to prove that M_0 is refined by M_1 . In fact, discharging proof obligations (either automatically or interactively) guarantees that each intermediate refined machine from M_1 to M_N ($M_0 \sqsubseteq M_1 \sqsubseteq \dots \sqsubseteq M_N$) are correct refinements of M_0 . As a consequence, the developed system is considered as a correct-by-construction model using the refinement approach.

We have used Event-B to specify the complex behaviour of OLSR protocol in a stepwise manner, from an abstract model to a more concrete specification. The main components and features of routing protocols as well as their crucial global properties are introduced in an abstract reusable model. This allows, on one hand, to use the abstract model to specify other routing protocols with common behaviour and on the other hand, to prove the generic global properties of protocols. Then, the concrete models refine the abstract specification while the global correctness of the protocol is preserved. The refined specification (model) can be reused in the development of other routing protocols. The refinement approach provides support for constructing reusable models that can be adapted to other routing protocols while the global correctness of the system is preserved.

Adding details into a system may significantly increase the states of the system, bearing the barrier of scalability. As a consequence, it is important to systematically categorise a model into several component models, for instance by decomposition. Event-B provides a form of decomposition, namely *modularisation*, in order to address complexity for the system under design. *Modularisation* in Event-B is based on *monotonicity* [6], and allows to study and refine each component model independently of others, facilitating scalability and reusability of component models. Each module is a component consists in a group of operations that can be called. During formal development, modules can be developed separately and later composed with the system [35].

4. Contributions

In this section, we describe the contribution of our original publications, building up this dissertation. We present the techniques applied in each paper for modelling and verifying communication protocols. We discuss the quantitative and qualitative analysis of protocols as well as provide a guideline on which formalism is more applicable in the context of WMN routing protocols. Our long term goal on providing a generic and reusable formal framework for modelling and verifying routing protocols is described as our last contribution in this dissertation. This study is carried out in order to provide *reliable* and *flexible* routing protocols, guaranteeing a certain number of requirements.

4.1 Paper I: Formal Analysis of Proactive, Distributed Routing

In this paper [38], we focus on modelling and analysing the functionality of a proactive routing protocol, OLSR, using Uppaal model checker. We formalise the main functionality of OLSR based on its English specification and define required properties of the protocol in mathematical formulas. The designed model based on the extended timed automata is an unambiguous and precise reflection of the OLSR core functionality that can be input for verification purposes. Our Uppaal model consists of a parallel composition of identical processes describing the behaviour of single nodes of the network. Each of these processes is itself a parallel composition of two timed automata, **Queue** and **OLSR**. The **Queue** automaton has been chosen to store incoming messages from other (directly connected) nodes. In other words, it denotes the input buffer of a node. The **OLSR** automaton models the complete behaviour of the routing protocol as described in the OLSR specification. It consists of 14 locations and 36 transitions precisely modelling the broadcasting and handling of the different types of messages.

The OLSR is verified in two settings: static networks and dynamic networks. In the context of static networks where no failure of links happens, we deal with three different properties (*reliability* analysis): establishing a

route from a source node to a destination node, delivering a data packet from a source node to its destination node, and finding optimal routes between nodes w.r.t. the number of hops. In the context of dynamic networks where there is a possibility of a random link failure, we focus on the recovery time of OLSR, to show how long it takes the OLSR protocol to recover (*flexibility* analysis). We verify our system for all network topologies up to five nodes for the two first properties mentioned earlier and sketch some malfunctioning of OLSR protocol as well as propose some modifications in order to overcome the problematic behaviour. We show that OLSR can not always deliver injected data packets via shortest paths when analysing OLSR functionality in static networks. Another malfunctioning of OLSR is concerned with its recovery time, meaning that OLSR needs a long period of time to be recovered after link failures even in small networks consisting of five nodes. This shows also the power of model checking technique in finding fundamental flaws of systems.

4.2 Paper II: Improved Recovery for Proactive, Distributed Routing

In this paper [40], we propose solutions to overcome the long recovery time of OLSR protocol by introducing a new type of message, namely ERROR message and modifying the OLSR w.r.t. timing constraints. We model and verify our proposed solution employing the Uppaal model checker in order to have a rigorous and precise analysis. We show that the recovery time of OLSR, due to link failures can be reduced to half. We illustrate our findings with a simpler formal model having a reduced state space and time for verification.

Our model containing our proposed modifications is verified in two settings: the static and dynamic networks. In the first case, we are interested in verifying the same properties verified for the original specification of OLSR described above to later compare how efficient our modifications are. Therefore, we check the route discovery and packet delivery properties, respectively, to verify whether or not the protocol establishes a route from a source node to a destination node as well as to verify if an injected packet to a source node is destined to its destination node (*reliability* analysis). In the second case, we deal with the recovery time of OLSR (*flexibility* analysis). We carry out the experiments in networks consisting of five nodes as for the model based on the original OLSR specification. We show that our solutions lead to a faster detection of broken links and to decreasing the recovery time of OLSR while the required properties of OLSR are preserved.

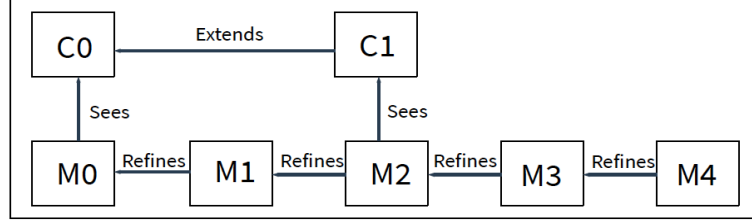


Figure 4.1: Our Event-B model of [41]

4.3 Paper III: Modelling Link State Routing in Event-B

In this paper [41], we focus on providing reusable models for routing protocols as well as verifying the OLSR protocol in larger networks using the Event-B formalism. We address the complexity of the OLSR protocol by dividing it into smaller components defined as contexts and machines in the system model. We start from an abstract model, introducing a simple routing process in which a single packet is sent between two directly connected nodes. In the next refinement steps, we add more details into the system model and make it stepwise more concrete. We verify our system requirements in each step before refining it to a more concrete model and we prove that the more concrete models refine the more abstract ones. Our model consists of five distinct layers of abstraction introducing five machines and two contexts (shown in Figure 4.1). In this figure, context **C1**, accessible by machine **M2**, extends context **C0**, accessible by machine **M0**. The most detailed (concrete) machine **M4** is the result of four refinement steps as: $M0 \sqsubseteq M1 \sqsubseteq M2 \sqsubseteq M3 \sqsubseteq M4$.

We design an understandable and manageable model that is generic for proactive routing in the first refinement steps and becomes more specific to OLSR protocol in the last refinements. Therefore, the first abstract models can be reused for modelling and verifying other routing protocols. In addition, we do not have the problem of scalability when modelling the system in Event-B since we define the number of nodes in the network as a parameter, e.g., n . We verified our system model for three main properties of OLSR: route establishment, packet delivery and optimal route finding (*reliability* and *flexibility* analysis). The first two properties, i.e., route establishment and packet delivery properties, were satisfied as it was expected (was shown in previous works above). The last property w.r.t. finding optimal routes cannot be proved suggesting that OLSR does not necessarily find the optimal route.

	Uppaal	Event-B
Specification language	Time automata, C-like language	Set theory, guarded commands language
Variable update	Transition: selection guard update	Event: parameter guard action
Modularity	Divided into several automata at the same level of abstraction	Divided into several machines at different levels of abstraction
Verification	CTL automatically providing counterexamples	First-order logic automatically and interactively no counterexamples
Scalability	Small-scale systems (finite)	Large-scale systems (infinite)
Real time	Precisely models timing variables (continuous time)	Partially models timing variables (discrete time)

Table 4.1: Overview of Uppaal and Event-B comparison

4.4 Paper IV: Uppaal vs Event-B for Modelling Optimised Link State Routing

In this paper [42], we carry out a general study on comparing both techniques that we used in our previous studies, i.e., Uppaal and Event-B, in order to determine which formalism can be employed when designing formal generic and reusable frameworks for routing protocols as our long term goal. First, we compare different models of OLSR protocol that we modelled and verified applying Uppaal and Event-B, and then we concentrate on describing different aspects of each formalism.

Our comparison w.r.t. the models deals with the following criteria: which parts of the protocol could be modelled, e.g., core functionality, which properties can be defined, what size of networks can be defined, and what data structures can be used to precisely express the behaviour of the protocol. Our study on comparing the employed techniques focuses on the specification language, updates of variables, modularity, verification (*reliability* and *flexibility* analysis), scalability, and real-time characteristic shown in Table 4.1. Our study provides a guideline on when to use each formalism for modelling and verifying communication protocols. In this paper, we point out on strengths and limitations of each technique in the context of routing protocols.

	loss=0%	loss=10%	loss=30%	loss=40%
DYMO	0.99	0.99	0.89	0.65
AODVv2-16	0.99	0.98	0.72	0.45

Table 4.2: Route establishment on 3x3 grid networks.

	loss 0%	loss 10%	loss 30%	loss=40%
DYMO	37.27	37.42	34.68	31.27
AODVv2-16	34.01	34.38	34.57	31.66

Table 4.3: Route quantity on 3x3 grid networks.

4.5 Paper V: AODVv2: Performance vs. Loop Freedom

In this paper [39], we concentrate on analysing and comparing two versions of the reactive routing protocol AODV in Uppaal SMC. The focus of our work is on Dynamic MANET On-demand (DYMO) and Ad-hoc On-demand Distance Vector version2 (AODVv2) routing protocols. We formalise the core functionality of both DYMO and AODVv2 protocols in Uppaal SMC for 3x3 grid topologies (9 nodes). While the model for AODVv2 is completely new, the model for DYMO is an improved version of those appearing in [31] and [21].

In both models, we consider a probabilistic model in order to model wireless communication behaviour, e.g., possible message loss and link failures (*flexibility* analysis). We compare the functionality of both protocols for four different properties: route discovery (Table 4.2), number of found routes (Table 4.3), optimal route finding (Table 4.4), and packet delivery (Table 4.5) (*reliability* analysis). Field **loss** in all tables represents the probability of losing messages in the network. Results show that the older DYMO protocol outperforms the most recent AODVv2 protocol, meaning that AODVv2 is paying the price of degraded performance compared to DYMO. We believe that it was the intention of protocol designers to accept the performance hit in order to keep AODVv2 as a loop-free protocol (Table 4.6).

	loss 0%	loss 10%	loss 30%	loss=40%
DYMO	0.94	0.84	0.67	0.48
AODVv2-16	0.95	0.86	0.58	0.37

Table 4.4: Optimal routing on 3x3 grid networks.

	loss 0%	loss 10%	loss 30%	loss=40%
DYMO	0.99	0.98	0.78	0.50
AODVv2-16	0.99	0.97	0.60	0.35

Table 4.5: Packet delivery on 3x3 grid networks.

	loss 0%	loss 10%	loss 30%	loss 40%
DYMO	1	2	2	2
AODVv2-16	0	0	0	0

Table 4.6: Number of loops in different configurations.

4.6 Paper VI: Adaptive Formal Framework for WMN Routing Protocols

In this paper [37], we present a generic, adaptive and reusable framework as well as crucial generic properties regarding routing protocol requirements, in order to precisely model and formally verify WMN routing protocols. Since formal modelling and verification of routing protocols emerge as challenging tasks and this stage of development is often skipped by the protocol designers, we put our effort on providing the adaptive framework. In this way, protocol designers can import the generic models and alter them based on protocol specifications and verify routing protocols prior to implementation. For these purposes, we employ Uppaal SMC for the following reasons: it overcomes the scalability problem of classical Uppaal, the limitation of Event-B for modelling timing aspects of protocols is not an issue in Uppaal SMC (it is possible to define timing constraints), it provides support for quantitative and qualitative analysis, and wireless communication can be simply modelled.

In our framework the main components defining a routing protocol are as following: communication between nodes, topology of the network and its changes (*flexibility* analysis), behaviour of each node (broadcasting, queueing, processing, discarding control messages, etc) and verification (shown in Figure 4.2). The behaviour of each node is also categorised into subcomponents: queue of a node that buffers messages, handler of a node that is responsible for tasks such as processing messages, storing information, etc, a generator which is unique to proactive protocols that broadcast control messages periodically, and timers that are used for the sleeping mode of nodes. The generic properties deal with the main requirements of WMN routing protocols. They are divided into three main properties w.r.t. the *reliability* analysis, e.g., if a route is established from a source node to a destination node (route establishment), how populated the routing tables of nodes are and how much information is provided in these tables (total knowledge of network), and if a data packet is delivered from a source node to its destination node (packet delivery). We show the applicability of our framework by modelling three routing protocols of WMNs, namely BATMAN, OLSR and

Communication		
<i>global</i>	<ul style="list-style-type: none"> Message type definitions Channels for each message type Meta variables for exchanging values 	
Nodes		Topology
Queue (proactive/reactive)		
<i>global</i>	<ul style="list-style-type: none"> Channel for synchronizing with <i>Handler</i> 	<i>global</i> <ul style="list-style-type: none"> Connectivity matrix Methods to add or delete connections Method to check whether nodes are connected
<i>local</i>	<ul style="list-style-type: none"> Buffer Methods for adding and deletion of messages 	<i>local</i> <ul style="list-style-type: none"> Clock for topology changes
<i>automaton</i>	<ul style="list-style-type: none"> Receiving messages Includes loss Passing to handler 	<i>automaton</i> <ul style="list-style-type: none"> Model for adding or removing connections
Timers (optional)		Verification
<i>global</i>	<ul style="list-style-type: none"> Flag for restart and expired timers Urgent restart channel 	<i>global</i> <ul style="list-style-type: none"> Variables for bookkeeping Methods for properties Methods to be used by tester
<i>local</i>	<ul style="list-style-type: none"> Boolean for running timer Array of clocks 	<i>local</i> <ul style="list-style-type: none"> Variables for bookkeeping Methods for properties
<i>automaton</i>	<ul style="list-style-type: none"> Invariant for running timer Restart Expiring timer 	<i>automaton</i> <ul style="list-style-type: none"> Test automaton
Handler (proactive/reactive)		
<i>global</i>	<ul style="list-style-type: none"> Channel for synchronizing with <i>Queue</i> Routingtable 	
<i>local</i>	<ul style="list-style-type: none"> Current message Clock for processing delay Methods update routing table Methods to create messages Methods to check whether message should be dropped. 	
<i>automaton</i>	<ul style="list-style-type: none"> Receiving from <i>Queue</i> Dropping messages Processing, and sending of new messages. 	
Generator (proactive)		
<i>automaton</i>	<ul style="list-style-type: none"> Generate control messages at regular intervals. 	

Figure 4.2: Generic structure of a WMN routing protocol model for verification, with respect to typical structure of an Uppaal model.

AODVv2 applied in 3x3 grid topologies. Similarity of the results from employing our framework with the results from the original models of protocols verifies the adaptability of our framework.

5. Related Approaches

In this section, we present existing literature that is relevant to this dissertation. We outline relevant formal approaches employed in order to specify, analyse and verify wireless networks and their routing protocols. The relevant formal techniques in this dissertation are categorised into three groups: model checking, statistical model checking and theorem proving techniques. Formal techniques allow both complex systems, and a set of requirements for these systems, to be modelled mathematically. The formal model of wireless networks and their routing protocols can then be rigorously checked against their formal specification. In the following, we present several studies that have been carried out using these techniques for the analysis of wireless networks and their protocols.

5.1 Model Checking

Model checking techniques have been used in several studies in the context of wireless networks. Fehnker et al. [24] applied the Uppaal model checker [10] to analyse qualitative properties of the AODV protocol [55] in all networks topologies containing up to 5 nodes. Their Uppaal model of the AODV protocol is based on a process-algebraic model [25], reflecting the specification of the protocol accurately and precisely, however abstracting the timing behaviour of AODV. They have reported on some problematic and unusual behaviour of the AODV protocol with the assistance of Uppaal model checker. Their study shows that protocols limitations can be discovered and their improved variants can be developed when applying model checking technique.

Chiyangwa and Kwiatkowska [16] applied Uppaal model checker to complement the other existing analysis of the AODV protocol. They modelled the AODV behaviour together with its timing aspects and investigated consequences of protocol parameters on the timing aspects of the protocol and verify system requirements such as timed route discovery and timed message delivery for a linear topology containing 14 nodes. Their results show that the lifetime of routes is dependent on the network size and they proposed a modification so that route timeouts can be adapted to network growth.

Chaudhary et al. [15] formally modelled the BATMAN routing protocol [48] applied in wireless mesh networks using the Uppaal model checker. Their formalisation revealed several inconsistencies and ambiguities in the corresponding BATMAN RFC. They focused on developing two models. The first model implements a literal reading of the BATMAN RFC which reflects the closest interpretation to the RFC whereas the second model focuses on BATMAN underlying concepts. They verified their basic untimed models in order to ensure loop-freedom, bidirectional link discovery, and route-discovery for a network consisting of 4 nodes located in a ring topology. They have also simulated a timed model of BATMAN in order to compare the performance and show that both models behave similarly when the time and number of required messages for route discovery matters. Based on their analysis, their alternative model improves the literal interpretation of the RFC by identifying significantly lower number of suboptimal routes.

Fehnker et al. [26] modelled and verified a medium access control protocol, the LMAC protocol [60], for wireless sensor networks employing Uppaal model checker. They have focused on systematically analyse all connected network topologies containing 4 and 5 nodes. Their study was conducted to detect and resolve collisions in such networks, providing valuable insight of the protocol. Results show that the protocol is not always able to detect collisions, i.e., no guarantee for collision detection for all scenarios. They have improved the protocol by decreasing the number of undetected collisions that prevent connection to the gateway.

The focus of the outlined studies is on verification of different routing protocols of wireless networks and shows the importance of protocol verification in this context. These studies also show that fundamental flaws in these systems can appear even in networks with small number of nodes using the model checking technique. The aim of this dissertation has been on the same line as these previous works, targeting the OLSR and AODVv2 protocols.

Steele and Andel [62] carried out a study of the OLSR protocol employing the model checker Spin [33]. They modelled the behaviour of the OLSR in Spin (Promela language) and then applied Linear Temporal Logic (LTL) to analyse the correct functionality of the protocol. They verified their system for the following properties: correct route discovery, correct relay selection, and loop freedom. Due to state space explosion, they analysed the network topologies up to 4 nodes. When taking symmetries into account they have analysed 17 topologies. Moreover, a timing analysis is not possible using Spin. Therefore, the model provided by Steele and Andel abstracts from timing; as we had shown, analysing OLSR with time variables reveals more shortcomings.

Liu et al. [43] presented a formal modelling framework for MANETs consisting of several mobility models together with wireless communication applying Real-Time Maude [52] (a formal specification for analysis of real-

time systems). They analysed the AODV protocol using their framework and their mobility models. Their framework mainly focuses on integrating a number of mobility models together with wireless communication.

5.2 Statistical Model Checking

Continuing this line of research in formal methods community, Höfner and McIver [31] made a comparison of the AODV and DYMO [56] (evolution of AODV) protocols on arbitrary networks up to 5 nodes applying Uppaal SMC [22]. They modelled and verified their systems considering perfect communication among nodes. Their analysis on five node networks shows that the more recent DYMO protocols has worse performance compared to older AODV protocol.

Fehnker et al. [23] proposed a topology-based model for mobility which abstracts from physical behaviour. They applied Uppaal SMC to model the mobile node that probabilistically changes the topology. Their proposed model covers the main features of the random waypoint and random walk mobility models. They have used their mobility model in combination with the AODV and LMAC protocols in order to carry out systematic analysis on these protocols in grid network topologies with respectively, 16 and 9 number of nodes.

These series of studies on the AODV routing protocol have been extended by adding a mobility model [30] in order to investigate the behaviour of AODV. They used Uppaal SMC to reason about AODV functionality in large and mobile scenarios containing 17 nodes (16 static nodes and 1 mobile node) located in a grid topology. Their study shows that some of the optional features of AODV are not useful and that the protocol encounters problematic and unexpected behaviour such as route discovery failure with a high probability.

Dal Corso et al. [21] studied the extended and generalised work done by [31]. They focused on 4x3 grids with the possibility of lossy communication. It means that the communication between nodes is defined as lossy and incoming messages can be lost in the reception of other nodes. They have employed Uppaal SMC to model the probability of message loss as well as to reason about more realistic network sizes. They showed contrary results compared to [31], indicating that DYMO is performing better compared to AODV in larger networks consisting of twelve nodes.

Fengling et al. [65] proposed a new method for analysing and evaluating wireless sensor network protocols employing statistical model checking. They have modelled wireless sensor network protocols as stochastic timed automata that deal with message loss and dynamic network topology (realistic wireless network characteristics). They studied the Timing-sync protocol for

wireless sensor networks having 3 to 20 number of nodes in order to show the feasibility and scalability of their method.

Battisti et al. [8] employed Uppaal SMC in order to investigate the robustness of gMAC protocol, a distributed clock synchronisation protocol for wireless sensor networks, in case of lossy communication. Their analysis is carried out on two settings based on classes of regular network topologies; 1) cliques: fully connected networks with arbitrary number of nodes and 2) small grids: same degree nodes with 25 number of nodes in a grid topology. Their results show that the probability that the protocol fails to synchronise the nodes is high when lossy communication is allowed in such networks.

The studies above present the power of statistical model checking in quantitative and qualitative analysis of large networked systems where probabilistic choices play important roles. We employed this formal technique in order to analyse AODVv2 and DYMO behaviour as well as to propose our generic framework for modelling, analysing and verifying network routing protocols.

5.3 Theorem Proving

There are several studies using theorem proving techniques in the context of wireless networks. Kamali et al. [36] applied refinement technique in order to formally model and analyse wireless sensor-actor networks. They have provided proofs showing that failed actor links can be temporarily replaced by communication via the sensor infrastructure, listing some assumptions. Applying refinement approach, they were able to prove that this recovery is correct and it terminates in a finite number of steps. They also generalised their formal development strategy in order to provide a reusable framework that can be applied for wider class of networks. They employed the Event-B formalisation based on the theorem proving technique and discharged their proof obligations using the RODIN platform (the integrated development framework for Event-B).

Méry and Singh [45] modelled the DSR protocol in a stepwise manner applying the Event-B formalism in order to analyse and reason about the behaviour of this protocol. Their refinement falls into 5 refinement steps starting from a very abstract model and gradually adding details into more concrete models. They have presented the system requirements w.r.t. the safety and liveness properties by defining them as invariants. Defined properties are established by proof of invariants, refinement of events, etc, using the Rodin platform.

Bhargavan et al. [13] employed the HOL theorem prover [28] together with the SPIN model checker [32] in order to analyse and verify properties of routing protocols in ad-hoc networks. They have focused on verifying the

AODV protocol, resulting in identifying errors in the specification of AODV that can cause a deadlock in the system.

Gawanmeh [27] applied Event-B formalism in order to model and verify the ZigBee protocol used for communication in wireless sensor networks. The protocol has been modelled at different levels of abstraction, precisely modelling the protocols primitives.

In order to model, verify and reason about wireless networks and their protocols with large number of nodes, the authors of previous papers have been employing the theorem proving technique. These papers show the importance of verification in the context of wireless networks and their protocols in order to ensure the correctness of such systems. We have continued this line of research by modelling the OLSR protocol in Event-B and ensuring its correctness in arbitrary networks with n number of nodes.

6. Discussion

In this section, we present the main achievements in this dissertation as well as outline our future research directions.

6.1 Summary

In this dissertation, we contributed at rigorously analysing and verifying different routing protocols of wireless mesh networks in order to ensure their reliability and flexibility. In case of unexpected behaviour or malfunctioning of the protocols, we proposed corresponding modifications to overcome those problems. As our long term goal, we developed an adaptive, generic and reusable formal framework for modelling as well as generic properties for verifying routing protocols of such networks. Our research approach consists of employing formal techniques in order to achieve our goals.

6.2 The Challenge

A very challenging task when studying wireless mesh network routing protocols is encountering prohibitively *complex* and often *ambiguous* specification. The complexity of the specification should be addressed while the model of the system still reflects the system behaviour correctly. The ambiguity of the specification should be addressed in the way that only one interpretation can be concluded after the model is built to be able to have a rigorous analysis of the system. Also in order to provide a generic and adaptive framework that can be reused when modelling and verifying other routing protocols, main characteristics and requirements of such networks should be precisely specified.

Another challenge is dealing with *scalability* as specification of protocols usually exceeds hundred pages. Modelling all protocol aspects in a detailed manner leads to models that are too large to be handled by automated verification. Therefore, the key challenge here is to deal with this problem of the “curse” of dimensionality. The protocol models require to be comprehensive enough in order to reflect the fundamental aspects of the protocols, the

aspects that are essential for the protocols to perform based on their specification. Often adding one more feature to the protocol models can make them unmanageable to analysis and verify—the so-called state-space explosion problem [18]. This is due to the exponential growth of the state space in the number of components and model variables. In order to overcome this challenge we applied abstraction.

6.3 What we have achieved

Our aim has been to develop a framework for analysing and verifying routing protocols of wireless mesh networks. The framework can be used by protocol designers to ensure the reliability and flexibility of protocols prior to deploying them in real life. Therefore, we provide:

1. A generic, adaptive and reusable formal development of such systems
2. Precise generic and reusable properties for rigorous quantitative and qualitative analysis

In order to achieve this aim, we have developed the following artefacts:

- A) Formal analysis of different categories of routing protocols to get insights from each protocol group, employing different formalisms ([38], [40], [41], [39]).
- B) A guideline on applicability of two formal frameworks (Uppaal and Event-B) that were applied for formal modelling and verification of protocols ([42]).
- C) A formal generic model for rigorous analysis of routing protocols ([37]).
- D) Methods and requirements for analysing wireless mesh network routing protocols ([37]).

Artefact (A), formal analysis of different categories of routing protocols, consists of providing formal models of different categories of protocols using different formalisms in order to formally analyse them and possibly improve them. This has been done to get insights about how different categories of protocols (reactive and proactive) function and to document main characteristics of these protocols which help us to achieve our goals (mentioned above (1) and (2)).

Artefact (B), a guideline on when to use different formal techniques (Uppaal and Event-B) in the context of wireless networks, presents pros and cons of each formalism when modelling and analysing network routing protocols, assisting us to choose appropriate formalism to achieve our goals (1) and (2).

Artefact (C), a formal generic model for rigorous analysis of routing protocols, contains high-level generic models in automata-based templates to specify wireless mesh network routing protocols, both reactive and proactive, that can be adapted based on a particular protocol. Formal development of the protocol will not start from scratch, but from applying the generic model. Therefore, this artefact is served as an answer to our goal (1).

Artefact (D), methods and requirements for analysing wireless mesh network routing protocols, provides the main routing protocol's requirements stated as properties in CTL and MITL syntaxes. These properties together with the adapted model are applied to rigorously analyse routing protocols quantitatively and qualitatively. So, this artefact is served as an answer to our goal (2).

6.4 Strengths and Limitations

In this dissertation, our main concerns were on how to deal with complexity and scalability when modelling the selected routing protocols. How should we model our systems in a way that they reflect the behaviour of the systems correctly while still keeping the models manageable to verify? The specifications of both routing protocols, OLSR and AODV, reach approximately hundred pages. Therefore, we had to *abstract away from the optional features* and only focused on modelling the *core functionality* of protocols, i.e., those functionalities that are necessary for the protocols to perform as intended by their specifications.

Using Uppaal, we were able to deal with the timing behaviour, wireless communication, and complex data structure of routing protocols. Also, the Uppaal GUI and Uppaal simulator provide a visualised interpretation of the system which makes the task of modelling easier. Since Uppaal carries out an exhaustive exploration of the state space of the model in order to guarantee that the system does not violate system requirements (properties), we were only able to analyse our models for *small networks* as the states space was exploding when having larger networks.

Event-B was an alternative to tackle the state space problem of Uppaal model checker (verifying the system for arbitrary number of nodes) and to manage the complexity of the routing protocols into distinct abstraction layers. However, Event-B does not automatically support the timing feature of protocols (it is possible to model the time only discretely). So, we *abstracted away from the timing aspect* when modelling our system in Event-B. Another limitation of both formalisms, i.e., Uppaal model checker and Event-B, is that we were not able to deal with *probability* analysis, e.g., message loss in the network.

We used Uppaal SMC as the substitution to overcome the state space

explosion of Uppaal model checker, timing issue of Event-B and probability issue of both formalisms. Using Uppaal SMC, we were able to model timing and probability for larger networks and carry out quantitative and qualitative analysis. However, Uppaal SMC does only give us a 99% guarantee of the results.

6.5 Future Directions

We have several possible directions to work on for the future. We have presented a generic framework for modelling wireless mesh networks routing protocols, and generic properties for analysing and verifying them. One possible research direction may consist of further extending of our current framework, adding other features of such protocols. For instance, we can analyse non-functional properties of wireless protocols, such as efficiency: expected energy consumption, etc. We will concentrate on two main aspects: battery depletion in wireless stations, and energy consumption within particular protocol mechanisms applying Probabilistic Timed Automata (PTA) [9] with prices [11]. PTA extend timed automata [4], the most successful formal model for timing-sensitive systems such as wireless protocols, with discrete probabilistic branching. In turn, PPTA extend PTA with costs that linearly grow with the residence time in states.

We are in particular interested in reasoning about unknown parameters in wireless systems. That is, the probabilities in the system do not necessarily need to be fixed, and can instead be given as expressions over some set of model parameters [20]. As an example, message loss in the network can be defined as a parameter. This is extremely useful if precise loss rates are not known, e.g., in early design stages. A message can be received with probability p and lost with probability $1 - p$, and now we are interested in calculating the probability values of p for which route discovery is successful in the network. This probability can be defined as a function in p . By inserting a suitable value for p , we obtain a concrete model without parameters. Parameter synthesis determines all parameter valuations where the probability of route discovery is more than a priori specified threshold. Parameter synthesis gives provable – and not just statistical – guarantees. By synthesising parameters fulfilling requirements optimally, we will also be able to optimise reliability, flexibility and efficiency e.g., minimising expected energy consumption.

7. Bibliography

- [1] Jean Raymond Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, 1996.
- [2] Jean Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
- [3] Jean Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, 12(6):447–466, 2010.
- [4] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- [5] Ralph-Johan Back and Reino Kurki-Suonio. Distributed cooperation with action systems. *ACM Transactions on Programming Languages and Systems*, 10(4):513–554, 1988.
- [6] Ralph-Johan Back and Kaisa Sere. From action systems to modular systems. In *Software - Concepts and Tools*, pages 1–25. Springer, 1994.
- [7] Ralph-Johan Back and J. Joakim Wright. *Refinement calculus - a systematic introduction*. Springer, 1999.
- [8] Luca Battisti, Damiano Macedonio, and Massimo Merro. Statistical model checking of a clock synchronization protocol for sensor networks. In Farhad Arbab and Marjan Sirjani, editors, *Fundamentals of Software Engineering*, pages 168–182. Springer Berlin Heidelberg, 2013.
- [9] Danièle Beauquier. On probabilistic timed automata. *Theoretical Computer Science*, 292(1):65 – 84, 2003.
- [10] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on Uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, volume 3185, pages 200–236. Springer, 2004.

- [11] Jasper Berendsen, David N. Jansen, and Frits W. Vaandrager. Fortuna: Model checking priced probabilistic timed automata. In *International Conference on the Quantitative Evaluation of Systems*, pages 273–281. IEEE, 2010.
- [12] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Größer, and Marcin Jurdzinski. Stochastic timed automata. *Logical Methods in Computer Science*, 10(4), 2014.
- [13] Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM*, 49(4):538–576, 2002.
- [14] Armin Biere, Edmund Clarke, Richard Raimi, and Yunshan Zhu. Verifying safety properties of a powerpc microprocessor using symbolic model checking without bdds. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, pages 60–71. Springer, 1999.
- [15] Kaylash Chaudhary, Ansgar Fehnker, and Vinay Mehta. Modelling, verification, and comparative performance analysis of the B.A.T.M.A.N. protocol. In Holger Hermanns and Peter Höfner, editors, *Workshop on Models for Formal Analysis of Real Systems*, volume 244, pages 53–65, 2017.
- [16] Sibusisiwe Chiyangwa and Marta Kwiatkowska. A timing analysis of aodv. In Martin Steffen and Gianluigi Zavattaro, editors, *Formal Methods for Open Object-Based Distributed Systems*, pages 306–321. Springer, 2005.
- [17] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 154–169. Springer, 2000.
- [18] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. *Model Checking and the State Explosion Problem*, pages 1–30. Springer, 2012.
- [19] Thomas. H Clausen and Philippe Jacquet. Optimized link state routing protocol (OLSR). RFC3626, 2003.
- [20] Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, and Ufuk Topcu. Synthesis in pMDPs: A tale of 1001 parameters. In *Automated Technology for Verification and Analysis*, pages 160–176. Springer, 2018.

- [21] Alice Dal Corso, Damiano Macedonio, and Massimo Merro. Statistical model checking of ad hoc routing protocols in lossy grid networks. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Method Symposium*, pages 112–126. Springer, 2015.
- [22] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.
- [23] Ansgar Fehnker, Peter Höfner, Maryam Kamali, and Vinay Mehta. Topology-based mobility models for wireless networks. In Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D’Argenio, editors, *Quantitative Evaluation of Systems*, pages 389–404. Springer, 2013.
- [24] Ansgar Fehnker, Rob van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann, and Wee Lum Tan. Automated analysis of AODV using UPPAAL. In Cormac Flanagan and Barbara König, editor, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214, pages 173–187. Springer, 2012.
- [25] Ansgar Fehnker, Rob van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann, and Wee Lum Tan. A process algebra for wireless mesh networks. In Helmut Seidl, editor, *European Symposium on Programming*, pages 295–315. Springer, 2012.
- [26] Ansgar Fehnker, Lodewijk van Hoesel, and Angelika Mader. Modelling and verification of the LMAC protocol for wireless sensor networks. In Jim Davies and Jeremy Gibbons, editors, *International conference on Integrated formal methods*, pages 253–272. Springer, 2007.
- [27] Amjad Gawanmeh. Embedding and verification of zigbee protocol stack in Event-B. *Procedia Computer Science*, 5:736 – 741, 2011.
- [28] Michael. J. C. Gordon and Thomas. F Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
- [29] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
- [30] Peter Höfner and Maryam Kamali. Quantitative analysis of AODV and its variants on dynamic topologies using statistical model checking. In Víctor Braberman and Laurent Fribourg, editors, *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 121–136. Springer, 2013.

- [31] Peter Höfner and Annabelle McIver. Statistical model checking of wireless mesh routing protocols. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *NASA Formal Method Symposium*, pages 322–336. Springer, 2013.
- [32] Gerard Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley Professional, 2003.
- [33] Gerard J. Holzmann. The model checker SPIN. *IEEE Transaction Software Engineering*, 23(5):279–295, 1997.
- [34] Yih-Chun Hu, Dave A. Maltz, and David B. Johnson. The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4. RFC 4728, 2007.
- [35] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilic, and Timo Latvala. Supporting reuse in event b development: modularisation approach. In Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Régine Laleau, and Steve Reeves, editors, *International conference on Abstract State Machines, Alloy, B and Z*, pages 174–188. Springer, 2010.
- [36] Maryam Kamali, Linas Laibinis, Luigia Petre, and Kaisa Sere. Formal development of wireless sensor-actor networks. *Science of Computer Programming*, 80, Part A(0):25 – 49, 2014.
- [37] Mojgan Kamali and Ansgar Fehnker. Adaptive formal framework for WMN routing protocols. In *Formal Aspects of Component Software*, pages 175–195. Springer, 2018.
- [38] Mojgan Kamali, Peter Höfner, Maryam Kamali, and Luigia Petre. Formal analysis of proactive, distributed routing. In *Software Engineering and Formal Methods*, pages 175–189. Springer, 2015.
- [39] Mojgan Kamali, Massimo Merro, and Alice Dal Corso. AODVv2: Performance vs. loop freedom. In *Theory and Practice of Computer Science*, pages 337–350. Springer, 2018.
- [40] Mojgan Kamali and Luigia Petre. Improved recovery for proactive, distributed routing. In *International Conference on Engineering of Complex Computer Systems*, pages 178–181. IEEE, 2015.
- [41] Mojgan Kamali and Luigia Petre. Modelling link state routing in Event-B. In *International Conference on Engineering of Complex Computer Systems*, pages 207–210. IEEE, 2016.

- [42] Mojgan Kamali and Luigia Petre. Uppaal vs event-b for modelling optimised link state routing. In Kamel Barkaoui, Hanifa Boucheneb, Ali Mili, and Sofiène Tahar, editors, *Verification and Evaluation of Computer and Communication Systems*, pages 189–203. Springer, 2017.
- [43] Si Liu, Peter Csaba Ölveczky, and José Meseguer. A framework for mobile ad hoc networks in Real-Time Maude. In Santiago Escobar, editor, *WRLA*, pages 162–177. Springer, 2014.
- [44] Si Liu, Peter Csaba Ölveczky, and José Meseguer. Formal analysis of leader election in MANETs using Real-Time Maude. In *Software, Services, and Systems*, pages 231–252, 2015.
- [45] Dominique Méry and Neeraj Kumar Singh. Analysis of DSR protocol in Event-B. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 401–415. Springer, 2011.
- [46] Robin Milner. *A Calculus of Communicating Systems*. Springer, 1982.
- [47] Robin Milner. *Communicating and mobile systems: the π -calculus*. Cambridge University Press, 1999.
- [48] Axel Neumann, Corinna Aichele, Marek Lindner, and Simon Wunderlich. Better approach to mobile ad-hoc networking (BATMAN). Internet draft00, 2008.
- [49] Peter Csaba Ölveczky. Towards formal modeling and analysis of networks of embedded medical devices in real-time maude. In *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 241–248, 2008.
- [50] Peter Csaba Ölveczky and Martin Grimeland. Formal analysis of time-dependent cryptographic protocols in Real-Time Maude. In *International Parallel and Distributed Processing Symposium*, pages 1–8, 2007.
- [51] Peter Csaba Ölveczky and Sigurd Meldal. Specification and prototyping of network protocols in rewriting logic. In *Proc. NIK*, 1998.
- [52] Peter Csaba Ölveczky and José Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 20(1):161–196, 2007.
- [53] Peter Csaba Ölveczky and Stian Thorvaldsen. Formal modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude. In *International Conference on Formal Methods for Open Object-Based Distributed Systems*, pages 122–140, 2007.

- [54] Doron Peled. Partial order reduction: Model-checking using representatives. In Wojciech Penczek and Andrzej Szalas, editors, *Mathematical Foundations of Computer Science 1996*, pages 93–112. Springer, 1996.
- [55] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, 2003.
- [56] Charles Perkins, Ratliff Stan, and John Dowdell. Dynamic manet on-demand (AODVv2) routing draft-ietf-manet-dymo. Internat drfat26, 2013.
- [57] Ashish Raniwala and Tzi cker Chiueh. Architecture and algorithms for an iee 802.11-based multi-channel wireless mesh network. In *Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 2223–2234 vol. 3, 2005.
- [58] Steve Schneider and Helen Treharne. CSP theorems for communicating B machines. *Formal Aspects of Computing*, 17(4):390–422, 2005.
- [59] Steve Schneider, Helen Treharne, and Heike Wehrheim. A CSP approach to control in event-B. In Dominique Méry and Stephan Merz, editors, *International conference on Integrated formal methods*, pages 260–274. Springer, 2010.
- [60] Chunju Shao, DENG Hui, Rajesh Pazhyannur, Farooq Bari, Rong Zhang, and S. Matsushima. IEEE 802.11 Medium Access Control (MAC) Profile for Control and Provisioning of Wireless Access Points (CAPWAP). RFC 7494, 2015.
- [61] J. Michael Spivey. *The Z notation: a reference manual*. Prentice-Hall, Inc., 1989.
- [62] Matthew F Steele and Todd R Andel. Modeling the optimized link-state routing protocol for verification. In *Symposium on Theory of Modeling and Simulation - DEVS Integrative*, pages 1–8. Society for Computer Simulation International, 2012.
- [63] Hunter Hanzhuo Wu, Aaron Gilchrist, Ky Sealy, Paul Israelsen, and Jeff Muhs. A review on inductive charging for electric vehicles. In *International Electric Machines Drives Conference (IEMDC)*, pages 143–147. IEEE, 2011.
- [64] Chor Low Yan Zhang and Jim Ng. Performance evaluation of routing protocols on the reference region group mobility model for MANET. *Wireless Sensor Network*, 3(5):92 – 105, 2011.

- [65] Fengling Zhang, Lei Bu, Linzhang Wang, Jianhua Zhao, Xin Chen, Tian Zhang, and Xuandong Li. Modeling and evaluation of wireless sensor network protocols by stochastic timed automata. *Electronic Notes in Theoretical Computer Science*, 296:261 – 277, 2013.

Complete List of Publications

1. Mojgan Kamali and Ansgar Fehnker, Adaptive Formal Framework for WMN Routing Protocols, In Kyungmin Bae and Peter Ölveczky (Eds.) *Proceedings of the 15th International Conference on Formal Aspects of Component Software (FACS 2018)*, Lecture Notes in Computer Science Vol. 11222, pp. 175-195, Springer, 2018.
2. Mojgan Kamali, Massimo Merro and Alice Dal Corso, AODVv2: Performance vs. Loop Freedom, In A Min Tjoa, et al. (Eds.) *Proceedings of the 44th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2018)*, Lecture Notes in Computer Science Vol. 10706, pp. 337-350, Springer, 2018.
3. Mojgan Kamali and Luigia Petre, Uppaal vs Event-B for Modelling Optimised Link State Routing, In Kamel Barkaoui et al. (Eds.) *Proceedings of the 11th International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS 2017)*, Lecture Notes in Computer Science Vol. 10466, pp. 189-203, Springer, 2017.
4. Mojgan Kamali and Luigia Petre, Modelling Link State Routing in Event-B, *In the proceedings of the 21st International Conference on Engineering of Complex Computer Systems (ICECCS 2016)*, IEEE proceeding, pp. 207-210, IEEE, 2016.
5. Mojgan Kamali and Luigia Petre, Comparing Routing Protocols, *In the proceedings of the 20th International Conference on Engineering of Complex Computer Systems (ICECCS 2015)*, IEEE proceeding, pp. 178 - 181, IEEE, 2015.
6. Mojgan Kamali and Luigia Petre, Improved Recovery for Proactive, Distributed Routing, *In the proceedings of the 20th International Conference on Engineering of Complex Computer Systems (ICECCS 2015)*, IEEE proceeding, pp. 206 - 209, IEEE, 2015.
7. Mojgan Kamali, Peter Höfner, Maryam Kamali and Luigia Petre, Formal Analysis of Proactive, Distributed Routing, In Radu Calinescu and

Bernhard Rumpe (Eds.) *Proceedings of the 13th International Conference on Software Engineering and Formal Methods (SEFM 2015)*, Lecture Notes in Computer Science Vol. 9276, pp. 175-189, Springer, 2015.

Part II

Original Publications

Paper I

Formal Analysis of Proactive, Distributed Routing

Mojgan Kamali, Peter Höfner, Maryam Kamali and Luigia Petre

Originally published in: *proceedings of the 13th International Conference on Software Engineering and Formal Methods (SEFM 2015)*, Lecture Notes in Computer Science Vol. 9276, pp. 175-189, Springer, 2015.

©2015 Springer. Reprinted with permission from the publisher.

Formal Analysis of Proactive, Distributed Routing

Mojgan Kamali^{1(✉)}, Peter Höfner^{2,3}, Maryam Kamali⁴, and Luigia Petre¹

¹ Åbo Akademi University, Turku, Finland
{mojgan.kamali,lpetre}@abo.fi

² NICTA, Sydney, Australia

³ University of New South Wales, Sydney, Australia

⁴ University of Liverpool, Liverpool, UK

Abstract. As (network) software is such an omnipresent component of contemporary mission-critical systems, formal analysis is required to provide the necessary certification or at least formal assurances for these systems. In this paper we focus on modelling and analysing the Optimised Link State Routing (OLSR) protocol, a distributed, proactive routing protocol. It is recognised as one of the standard ad-hoc routing protocols for Wireless Mesh Networks (WMNs). WMNs are instrumental in critical systems, such as emergency response networks and smart electrical grids. We use the model checker Uppaal for analysing safety properties of OLSR as well as to point out a case of OLSR malfunctioning.

1 Introduction

Routing is at the centre of network communication, which in turn, is part of the backbone for numerous safety-critical systems. Examples are networks for telecommunication systems, for emergency response, or for electrical smart grids. In these and other examples, the communication is often truly distributed, without depending on any central entity (router) for coordination. Another important characteristics of these networks is that the network topology can change: in the case of emergency networks nodes might just fail; in case of telecommunication systems nodes such as laptops and mobile phones can move within the network, and even enter or leave a network. In this paper we focus on distributed routing mechanisms in such wireless networks; due to their wide-spread usage in critical systems, we aim at a formal model, which paves the way for a formal analysis.

A routing protocol enables node communication in a network by disseminating information enabling the selection of routes. In this way, nodes are able to send data packets to arbitrary (previously unknown) destinations in the network. Shortcomings in the routing protocol immediately decrease the performance and reliability of the entire network. Due to the possibility of topology changes information has to constantly be updated to maintain the latest routing information within the network. In this paper we focus on such self-organising wireless multi-hop networks which provide support for communication without relying on a wired infrastructure. They bear the benefit of rapid and low-cost

network deployment. The Optimised Link State Routing (OLSR) protocol [4], a proactive routing protocol, is identified as one of the standard routing protocol for Wireless Mesh Networks (WMNs) by the IETF MANET working group.¹ By distributing control messages throughout the network, proactive protocols maintain a list of all destinations together with routes to them.

Traditionally, common methods used to evaluate and validate network protocols are test-bed experiments and simulation in ‘living lab’ conditions. Such an analysis is usually limited to very few topologies [7]. In such experiments not only the routing protocol is simulated, but also all other layers of the network stack. When a shortcoming is found, it is therefore often unclear whether the limitation is a consequence of the routing protocol chosen, or of another layer, such as the underlying link layer. In this paper, we abstract from the underlying link layer; hence a shortcoming found is definitely a problem of the routing protocol.

Another problem with specifications in general and with the description of OLSR in particular is that specifications are usually given in English prose. Although this makes them easy to understand, it is well known that textual descriptions contain ambiguities, contradictions and often lack specific details. As a consequence, this might yield different interpretations of the same specification and to different implementations [9]. In the worst case, implementations of the same routing protocol are incompatible.

One approach to address these problems is using formal methods in general and model checking in particular. Formal methods provide valuable tools for the design, evaluation, and verification of WMN routing protocols; they complement alternatives such as test-bed experiments and simulation. These methods have a great potential on improving the correctness and precision of design and development, as they produce reliable results. Formal methods allow the formal specification of routing protocols and the verification of the desired behaviour by applying mathematics and logics [3]. In this way, stronger and more general assurances about protocol behaviour can be achieved.

In this paper we present a concise and unambiguous model for the OLSR protocol. The model is based on extended timed automata as they are used by the model checker Uppaal. As a consequence we report also on results of applying model checking techniques to explore the behaviour of OLSR. Model checking (e.g. [3]) is a powerful approach used for validating key correctness properties in finite representations of a formal system model.

The paper is structured as follows: in Sect. 2, we overview the OLSR protocol and in Sect. 3 we shortly discuss the Uppaal model of OLSR based on RFC 3626 [4]. Section 4 is the core of our paper where we present the results of our analysis. We review related work in Sect. 5 and propose future research directions in Sect. 6.

2 Optimized Link State Routing—An Overview

The Optimised Link State Routing (OLSR) protocol [4] is a proactive routing protocol particularly designed for Wireless Mesh Networks (WMNs) and Mobile

¹ <http://datatracker.ietf.org/wg/manet/charter/>.

Ad hoc Networks (MANETs). The proactive nature of OLSR implies the benefit of having the routes available at time needed. The underlying mechanism of this protocol consists in the periodic exchange of messages to establish routes to previously unknown destinations, and to update routing information about known destinations. OLSR works in a completely distributed manner without depending on any central entity. The protocol minimises flooding of control messages in the network by selecting so-called Multipoint Relays (MPRs). Informally, an MPR takes over the communication for a set of nodes that are one-hop neighbours of this node; these one-hop neighbours receive all the routing information from the MPRs and hence do not need to send and receive routing information from other parts of the network.

Nodes running OLSR are not restricted to any kind of start-up synchronisation. Every node broadcasts a *HELLO message* every 2s and detects its direct neighbour nodes by receiving these messages. Since HELLO messages contain information about all one-hop neighbours of the originator, receiving nodes can establish routes to their two-hop neighbours, too. HELLO messages traverse only one wireless link (a single hop), and are not forwarded by any node.

After receiving HELLO messages from direct neighbours, every node selects a particular one-hop neighbour, its MPR, and selected MPRs are aware of those nodes that have selected them as an MPR. MPRs broadcast *Topology Control (TC) messages* every 5s to build and update topological information. These messages are retransmitted (forwarded) through the entire network by MPRs. This means that if a node is not an MPR and receives TC messages, it processes those messages, but will not forward them. Every TC message contains the routing information provided by the originator. While receiving control messages from other nodes, every node updates its routing table according to the information received. After broadcasting and forwarding control messages via nodes, routes to all reachable destinations should be established by all nodes. Nodes can use the established routes to send data packets through the network.

Information stemming from HELLO messages is considered valid for 6s (three times the interval between sending HELLO messages); information from TC messages for 15s (three times the interval between sending TC messages). Routing table entries are marked as invalid if these times have passed.

More details about OLSR can be found in its specification [4]; a concrete example of OLSR running on a topology of 5 nodes can be found in [13].

3 Modelling OLSR in Uppaal

Uppaal [1, 15] is a well-established model checker for modelling, simulating and verifying real-time systems. It is designed for systems that can be modelled as networks of (extended) timed automata. We use Uppaal for the following reasons: (i) it provides two synchronisation mechanisms: broadcast and binary synchronisation; (ii) it provides common data structures, such as structs and arrays, and a C-like programming language—these features are used to model routing tables and update-operations on such tables; last, but not least, (iii) Uppaal

provides mechanisms and tools for considering timed variables—this is needed since OLSR highly depends on on-time broadcasting of control messages. In the remainder, we describe Uppaal to the extent needed in this paper.

3.1 Uppaal’s Timed Automata

The modelling language of Uppaal extends timed automata with various features, such as types and data structures [1]. A system state is defined as the value of all local and global variables. Every automaton can be presented as a graph with locations (nodes) and edges between these locations together with guards, clock constraints, updates and invariants. Clocks are variables that evaluate to real numbers and that are used in order to measure the time progression.

Each location may have an invariant, and each edge may have a guard, a synchronisation label, and/or an update of some variables. Guards on transitions are used to restrict the availability (enabledness) of transitions. Synchronisation happens via channels; for every channel a there is one label $a!$ to identify the sender, and $a?$ to identify receivers. Transitions without a label are internal; all other transitions use one of the two following types of synchronisation [1].

In *binary handshake* synchronisation, one automaton that has an edge with a $!$ -label synchronises with another automaton with the edge having a $?$ -label. These two transitions synchronise only when both guards hold in the current state. When the transition is taken, both locations will change, and the updates on transitions will be applied to the variables; first the updates will be done on the $!$ -edge, then the updates occur on the $?$ -edge. When having more than one possible pair, the transition is selected non-deterministically [1].

In *broadcast* synchronisation, one automaton with an $!$ -edge synchronises with several other automata that all have an edge with a relevant $?$ -label. The initiating automaton is able to change its location, and apply its update if and only if the guard on its edge is satisfied. It does not need a second automaton to synchronise with. Matching $?$ -edge automata must synchronise if their guards evaluate to true in the current state. They will change their location and update their states. First the automaton with the $!$ -edge updates its state, then the other automata follow. When more than one automaton can initiate a transition on an $!$ -edge, the process of choosing occurs non-deterministically [1].

Uppaal’s verifier uses Computation Tree Logic (CTL) (e.g. [6]) to model system properties. CTL offers two types of formulas: state formulas and path formulas. State formulas describe individual states of the model, while path formulas quantify over paths in the model. A path contains an (infinite) sequence of states. In this paper we employ the path quantifier **A** and the temporal operator **G**. **A** ϕ means that the formula ϕ holds for all paths starting from the current state. **G** ϕ means all future states (including the current one) satisfy ϕ . Formulas combine the path quantifiers and the temporal operators, e.g. **AG** ϕ holds if ϕ holds on all states in all paths originating from the current state. This is also denoted as **A**[] ϕ in Uppaal [1].

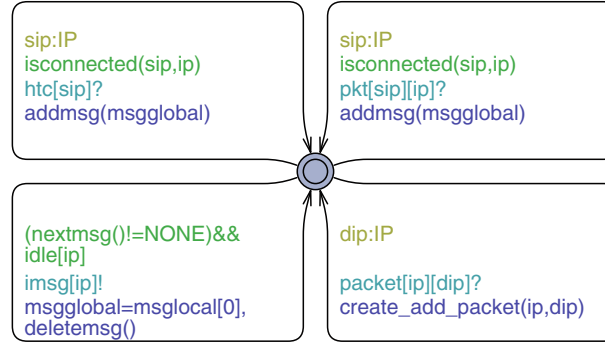


Fig. 1. The Queue automaton.

3.2 A Uppaal Model of OLSR

We now present an overview of our OLSR model. The model is described in detail in [13] and can be downloaded at hoefner-online.de/sefm15/. We model OLSR in Uppaal as a parallel composition of identical processes describing the behaviour of single nodes of the network. Each of these processes is itself a parallel composition of two timed automata, **Queue** and **OLSR**.

The **Queue** automaton (depicted in Fig. 1) has been chosen to store incoming messages from other (directly connected) nodes. In other words, it denotes the input buffer of a node. The received messages are buffered and then, in turn, send to the **OLSR** automaton for processing. Both actions on the top of Fig. 1 receive messages from other nodes in the network while the action on the lower right of Fig. 1 receives data messages from the same node. Messages can only be received if a node `ip` is connected to the sender `sip`. The channel `htc[sip]` receives a broadcasted message (HELLO or TC) from `sip` and stores the message to a local data queue, using the function `addmsg`. Both `pkt` and `packet` are handshake synchronisations and handle data messages travelling through the network and new messages injected by a client, respectively. Whenever the message-handling routine **OLSR** is ready to handle a message (`idle[ip]`), a message is moved from the message queue to **OLSR**, using the channel `imsg`.

OLSR models the complete behaviour of the routing protocol as described in [4]. It consists of 14 locations and 36 transitions precisely modelling the broadcasting and handling of the different types of messages. **OLSR** is busy while sending messages, and can accept a new message from **Queue** only once it has completely finished handling a message. Whenever it is not processing a message and there are messages stored in **Queue**, **Queue** and **OLSR** synchronise on the channel `imsg[ip]`, transferring the relevant data from **Queue** to **OLSR**. The automaton uses a local data structure to model the routing table of a node. Routing tables provide all information required for delivering packets. A routing table `rt` is an array of entries, one entry for each possible destination. An entry is modelled by the data type `rtentry`:

```

typedef struct {
    IP dip; //destination address
    int hops; //distance (number of hops) to the destination dip
    IP nhopip; //next hop address along the path to the destination dip
    SQN dsn; //destination dip sequence number
} rentry;

```

IP denotes a data type for all addresses and SQN a data type for sequence numbers. OLSR uses sequence numbers to check whether received messages are new or have already been processed. In our model, integers are used for these types.

The predicate `isconnected[i][j]` denotes a node-to-node communication, i.e., the nodes are in transmission range of each other. Communication between nodes happens via channels. The broadcast channel `htc[ip]` models the propagation of HELLO and TC messages where a message can be received by all one-hop neighbours. Each node has a broadcast channel, and every node in the range may synchronise on this channel. We also use the binary channel `packet[i][j]` to model the unicast sending of a data packet from `i` to `j`; this packet is generated by the user layer.

To model rigorous timing behaviour, we define 3 different clocks for every OLSR automaton: `t_hello` and `t_tc` are used to model on-time broadcasting HELLO and TC messages, and `t_send` models the time consumption for sending messages. According to the specification of OLSR, Hello messages are sent every 2000 ms. Considering a sending time of 500 ms (in our model `time_sending` = 500), nodes have to broadcast a new message 1500 ms after the last message was successfully distributed. For each OLSR automaton, we use two clock arrays `t_reset_rt` and `t_reset_rt_topo` of size N (the number of nodes in the network) to indicate the expiry time of one-hop and two-hop neighbours, and the expiry time of nodes which are more than two hops away, respectively.

To provide a realistic network set up, we model each node to send its first HELLO message non-deterministically between $[0, \text{time_between_hello})$. Afterwards, whenever `t_hello` reaches `time_between_hello`, OLSR resets `t_hello` and `t_send` to 0 before the HELLO message is broadcast.

Nodes receiving a HELLO message, update their routing tables for the originator of the message, learn about their two-hop neighbours and select their MPRs and MPR selectors using the functions `updatehello`, `updatetwohop` and `setmpr`, respectively. Furthermore, `t_reset_rt` is reset for originator of the message and its one-hop neighbours, which shows that new information has been received and this information is valid for 6000 ms.

After MPR nodes have been selected, each of them prepares for broadcasting TC messages to the connected nodes. TC messages are sent every 5000 ms. When `t_tc` reaches `time_between_tc`, `t_tc` and `t_send` are reset to 0. Then, a TC message is generated by `createtc` function and is broadcast to other nodes.

While transferring a TC message from Queue, `t_reset_rt_topo` is reset to 0 for the originator of the message and its MPR selectors, and if the message is considered for processing, the routing table is updated for the TC generator and its MPR selectors, using `updatetc` and `updatemprselector` functions, respectively.

If the receiver is an MPR then the TC messages can be forwarded. Forwarding messages also takes time in our model, namely `time_sending`. We note that OLSR might have to broadcast different messages at the same time. As an example, at some point a HELLO, a TC and maybe a TC to be forwarded are supposed to be broadcast; the sending time `time_sending` is counted only once and these messages are broadcast simultaneously. We consider this behaviour in our model as well. The full model, showing all details, is available online at hoefner-online.de/sefm15/.

4 Analysis

We analyse properties of the OLSR protocol in two different settings. First, we assume static network topologies, and then we allow changes in the network. The first series of experiments focuses on three properties:

- (1) *route establishment* for all topologies up to 5 nodes;
- (2) *packet delivery* in all topologies up to 5 nodes;
- (3) *route optimality* in topologies of up to 7 nodes.

We will show that OLSR does not always find optimal routes and propose a modification of OLSR that addresses this problem.

For the second series of experiments we assume dynamic network topologies where an arbitrary link fails. We focus on another property:

- (4) the *route discovery time*, i.e., we investigate the time during which there is no guaranteed packet delivery.

After analysing the route discovery time, we propose a modification that shortens this time; this modification will be analysed as well (Property (5)).

Due to the proactive nature of OLSR, our Uppaal model is pretty complex and contains several clocks, next to a complex data structure. As a consequence, state space explosion is a problem for our experiments. To address this problem, we apply different techniques supported by Uppaal to minimise the state space of our system model [5, 16, 17]. In particular, our model makes use of priority channels. By this we can order ‘internal actions’, i.e., actions that are running on a single node, and that are independent of other nodes and hence the order of the actions does not matter. For Properties (1), (2), (4) and (5), we give the highest priority to channels of node `a1` and the lowest priority to channels of node `a5`.

We also take into account symmetries of topologies, i.e., in case two topologies are isomorphic (up to renaming of nodes), we only analyse one. As a consequence we can reduce the number of experiments, by assuming, for Properties (1)–(5), that the originator is always the same node, denoted by `OIP1`, and the destination is always `DIP1`.

For the experiments we use the following set up: 3.2 GHz Intel Core i5, with 8 GB memory, running the Mac OS X 10.9.5 operating system. For all experiments we use Uppaal 4.0.13.

4.1 Static Topologies

Set Up. In this first series of experiments, we define another automaton, called **Tester1**, which injects a data packet to OIP1 to be delivered at destination DIP1. It is depicted in Fig. 2. It provides a local clock `clk`, which is used for invariants and guards. The location-invariant `clk ≤ 3*time_between_tc` in combination with the transition-guard `clk ≥ 3*time_between_tc` guarantees that the packet is injected at time point `3*time_between_tc`; hence a couple of control messages (HELLO and TC) have already been sent and most of the routes should have been established. The packet is injected to node OIP1 via the channel `packet[OIP1][DIP1]`.

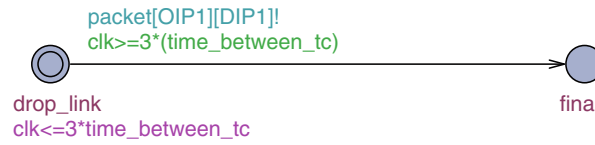


Fig. 2. The **Tester1** automaton.

The first property we are going to analyse (Property 1) is route establishment. It states that if the packet has been injected (**Tester1** is in location **final**), and all messages have been handled by all nodes (`emptybuffers()`) then OLSR has established a route between OIP1 and DIP1. This safety property using the Uppaal syntax is expressed as

$$\begin{aligned} A[] \quad & ((\text{Tester1.final} \ \&\& \ \text{emptybuffers}()) \ \text{imply} \\ & \text{node(OIP1).rt[DIP1].nhopip} \neq 0) \end{aligned} \quad (1)$$

Remember that the CTL formula $A[]\phi$ is satisfied iff ϕ holds on all states along all paths. The variable `node(OIP1).rt` represents the routing table of the originator node OIP1 and `node(OIP1).rt[DIP1].nhopip` expresses the next hop for the destination DIP1; if the next hop is not 0 a route is established.

The second property, packet delivery, is that if a packet is injected to the system, it is eventually delivered to the destination DIP1. In Uppaal this can be expressed as

$$\begin{aligned} A[] \quad & ((\text{Tester1.final} \ \&\& \ \text{emptybuffers}()) \ \text{imply} \\ & \text{node(DIP1).delivered} \neq 0) \end{aligned} \quad (2)$$

Here, `node(DIP1).delivered` indicates whether the injected data packet is received by the destination DIP1. Property 2 is stronger than Property 1 in the sense that the route is not only established, but it must be correct and used. Moreover this property implies loop freedom of OLSR, meaning that no packet is sent in cycles forever, without ever reaching the final destination.

The first two experiments are performed for all topologies up to five nodes, up to isomorphism and renaming. There are 444 of such topologies.

The third property, route optimality, checks if OLSR establishes optimal routes, after broadcasting, forwarding and processing TC messages. In our experiments we measure optimality with regards to shortest routes. Since we have full control over the topologies we are running the experiments with, we can determine the shortest possible route. We investigate this property for a ring topology of 7 nodes, as shown in Table 1.² Property 3 is expressed as

$$\begin{aligned} A[] \quad & ((\text{Tester1.final} \ \&\& \ \text{node}(\text{OIP1}).a \neq 0) \ \text{imply} \\ & \text{node}(\text{OIP1}).rt[\text{DIP1}].hops == 3) \end{aligned} \quad (3)$$

Here, $\text{node}(\text{OIP1}).a \neq 0$ indicates whether OIP1 has sent its packet to the next node along the path to DIP1; $\text{node}(\text{OIP1}).rt[\text{DIP1}].hops$ shows the number of hops from the originator OIP1 to the destination DIP1 which must be equal to 3. We also checked Property 3 on all topologies up to 5 nodes. The results, however, are not of real interest, since not much can go wrong w.r.t. shortest routes. As a consequence we picked topologies of size 7 to analyse route optimality.

Results. To analyse and verify OLSR, we evaluate Properties (1) and (2) in all network topologies up to 5 nodes. Property (1) is satisfied for all these networks: when the `Tester1` is in location `final`, node `OIP1` has established a route to node `DIP1`. This property confirms the propagation of HELLO and TC messages and also the correctness of the MPR selection mechanism. Hence, node `OIP1` is ready to send data packets to node `DIP1`.

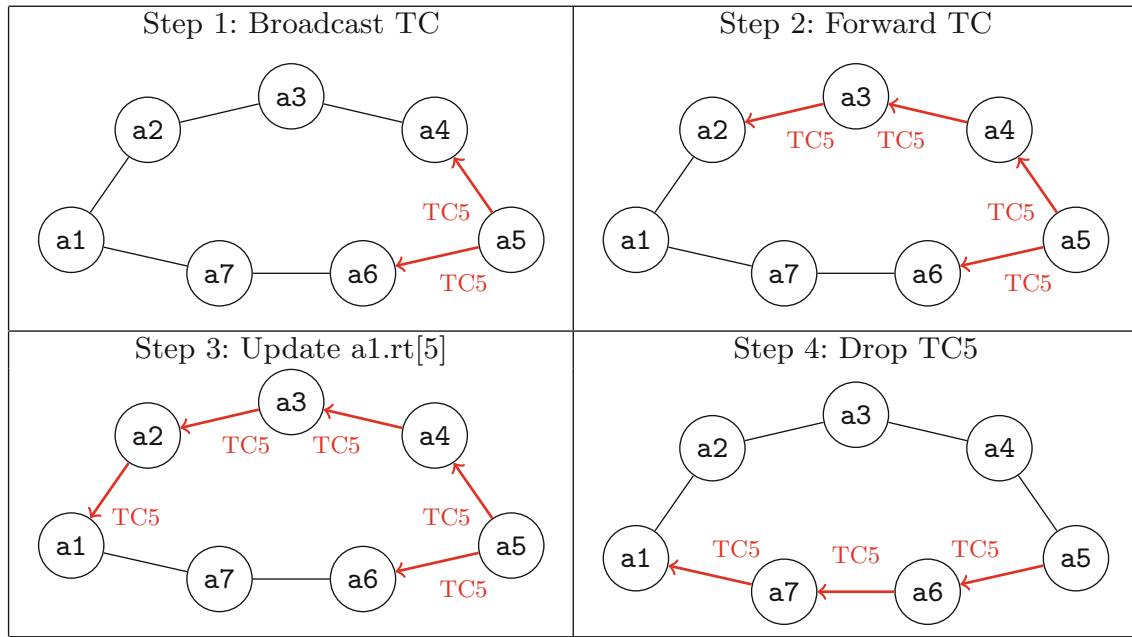
As mentioned before, Property (2) is stronger than Property (1). It models that all nodes have the information about all other nodes in the network, to deliver their data packets. In theory, the originator node `OIP1` could have a routing table entry for the destination node `DIP1`, stating that it should send a packet to its immediate next neighbour along the path to the destination `DIP1`; the next node itself might have no information about the destination `DIP1`, so all packets for the destination `DIP1` stemming from the originator `OIP1` would be lost. However, Property (2) is also satisfied for all topology up to size 5: all nodes have updated their routing tables in the network; therefore, they are able to deliver data packets to the arbitrary destination node `DIP1`.

While performing the analysis of Properties (1) and (2), we also performed some statistics: the Uppaal verifier analysed in average 1868996 states for each experiment; the largest one has 5314328 states, and the median is 1688368. Exploring these state spaces took on average 56 min.

Property (3), which analysis route optimality in topologies of size 7, is not satisfied. This proves that OLSR is not always able to find optimal routes.

Table 1 illustrates this phenomenon with an example found by Uppaal. In this example, `Tester1` synchronises with the `Queue` of node `a1`, which is the originator `OIP1` of the packet. The packet is intended for node `a5`. At some point, `a5` broadcasts a TC message (here indicated by `TC5`) to its neighbours `a4` and `a6`. While `a4` forwards the message to `a3`, `a6` is busy working on other

² There are too many topologies of that size, so we cannot analyse all topologies.

Table 1. Establishment of non-optimal routes in a 7 node topology

stuff and the message is kept in the message queue of **a6**. The TC message is forwarded subsequently via nodes **a3** and **a2** (Table 1: Step 2). As a consequence, node **a1**, updates its routing table entry for node **a5** (Table 1: Step 3). When **a1** receives TC5 via node **a7**, it has already updated its table for this node, and drops this message, since it has seen TC5 before. (Table 1: Step 4). By dropping this message **a1** misses out the chance TC5 to establish a shorter route. Similar examples are found for other routing protocols for WMNs [18].

Dropping a message with the same sequence number follows the specification:

“if there exists a tuple in the duplicate set, where:
D_addr == Originator Address,
AND
D_seq_num == Message Sequence Number
then the message has already been completely processed and MUST not be processed again.” [RFC3626, page 17]

This text snippet, copied from the RFC, shows that our model reflects the intention of OLSR; any message which is received and has already been handled (same sequence number) should be dropped. The idea is that the first message received must have travelled via the optimal path, which is not the case. A simple solution to this problem is to compare the potentially new route versus the routing table, in case the sequence numbers are the same. To reduce message flooding the message is only forwarded if the routing table is updated, i.e., if the hop count is strictly smaller.

4.2 Dynamic Topologies

Set Up. In the second series of experiments, we investigate the behaviour of OLSR after an arbitrary link is removed. Removing a link reflects a change in the topology. We define an automaton, called **Tester2** and depicted in Fig. 3, which drops the link between the two nodes `id_1` and `id_2`. We assume that the link breaks after $3 \cdot (\text{time_between_tc} + \text{time_sending})$ (in our model at 15000 ms), a time when all nodes have received information about all other nodes in the network (all routing tables have been updated for all nodes). Upon link breakage there is no connectivity between these two nodes; yet, each of them has the information about the other one. The packet, which should be sent from OIP1 to DIP1 is injected later on. By this we can analyse how quickly OLSR recovers from topology changes.

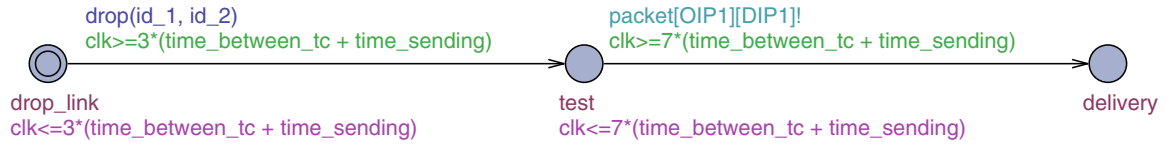


Fig. 3. The **Tester2** automaton.

Based on RFC 3626 (see box below), the information about one-hop and two-hop neighbours of a node is valid for $3 \cdot \text{REFRESH_INTERVAL}$, which equals 6000 ms; information about nodes which are more than two hops away from that node is valid for $3 \cdot \text{TC_INTERVAL}$, that equals 15000 ms.

“NEIGHB_HOLD_TIME = $3 \cdot \text{REFRESH_INTERVAL}$
 TOP_HOLD_TIME = $3 \cdot \text{TC_INTERVAL}$ ” [RFC3626, page 64]

This means information about one-hop and two-hop neighbours of a node is not available any longer if their corresponding clocks in the routing table have not been refreshed during 6000 ms; this indicates the breakage of a link. Also, if a node has not received TC messages from other MPR nodes for more than 15 s, information about those nodes is removed from the table.

We consider one desirable property of this protocol which indicates whether or not the injected packet is delivered at the destination if one link has been removed. In Uppaal syntax this safety property can be expressed as

$$A[] ((\text{Tester2.delivery} \ \&\& \ \text{emptybuffers}()) \ \text{imply} \ \text{node(DIP1).delivered} \neq 0) \quad (4)$$

After the topology has been changed and the packet has been injected, the automaton **Tester2** is in location **delivery**. If then the message buffers are empty (similar to the experiments described before) then we check if the packet has been delivered. (`node(DIP1).delivered != 0`).

Results. Property (4) is only satisfied for those topologies up to 5 nodes where the dropped link is not critical. In our model, a link is said to be critical if after link breakage there is no other link from that node to the other nodes along the path to the destination to be substituted with the broken one.

This experiment shows that the recovery in these topologies takes around 20 s (between 15000–35000 ms), which is a long period; in particular since we only consider networks of small size. As a consequence, this means that only after 35 s, the packet can certainly be delivered. The reasons for this long period are as following:

- After a link break occurred, some nodes might broadcast control messages (HELLO or TC) with incorrect (old) information, since nodes have not reset their tables for those nodes affected by link breakage. Based on RFC 3626, nodes reset their tables for the nodes from whom no control message is received after 6 and 15 s, respectively.
- At the time a link breaks, there are usually messages in the queue which need to be processed. These messages contain again out-dated information. So, the routing table is updated for the originator and one-hop neighbours of the message when receiving a HELLO, and for the originator and MPR selectors of the messages originator upon receiving a TC, even if the link does not exist anymore.
- Even when some nodes learn about the link breakage and reset the corresponding information in the routing table, it needs time to distribute this new knowledge.

Modifications. A solution to decrease the long recovery time of OLSR is to reduce NEIGHB_HOLD_TIME and TOP_HOLD_TIME to 2*REFRESH_INTERVAL and 2*TC_INTERVAL, respectively. To verify our proposal, we consider Property (5). This property states that refreshing routing tables in our proposed timing helps to reduce the recovery time.

$$\begin{aligned} A[] \quad & ((\text{Tester3.delivery} \ \&\& \ \text{emptybuffers}()) \ \text{imply} \\ & \text{node(DIP1).delivered} \neq 0) \end{aligned} \quad (5)$$

Similarly as for Property (4), Property (5) is satisfied for all topologies up to 5 nodes where the dropped link is not a critical link. After 25000 ms, the packet is definitely delivered at the destination. Therefore, it is feasible to reduce the recovery time of OLSR about 10000 ms (the difference between 35000 and 25000) using our proposed timing.

An alternative solution would be the introduction of error messages. As soon as a link break is identified, an error message should be sent to MPRs to inform the nodes and to correct the information in the routing tables as soon as possible. This modification would be in the same spirit as error messages used for other routing protocols, such as the AODV routing protocol. However, the analysis of this improvement is left for future work.

5 Related Work

While modelling and verifying protocols is not a new research topic, attempts to analyse routing protocols for dynamic networks are still rather new and remain a challenging task. Model checking techniques have been applied to analyse protocols for decades, but there are only a few papers that use these techniques in the context of mobile ad-hoc networks, e.g. [2]. In the area of WMNs, Uppaal has been used to model and analyse the routing protocols AODV and DYMO, see [7, 8, 10]. However, to the best of our knowledge, our study is the first aiming at a formal model of OLSR core functionality considering time variables.

Clausen et al. [4] specify the OLSR protocol in English prose. This paper is the official description currently standardised by the IETF. Jacquet et al. [12] also provide a high-level description of OLSR describing the advantages of this protocol, when compared to the others. However, none of these papers provide a formal model or a formal analysis of the protocol.

Steele and Andel [20] provide a study of OLSR using the model checker Spin [11]. They design a model of OLSR in which Linear Temporal Logic (LTL) is used to analyse the correct functionality of this protocol. They verify their system for correct route discovery, correct relay selection, and loop freedom. Due to state space explosion their analysis is limited to four node topologies only. When taking symmetries into account they analyse 17 topologies. Moreover, a timing analysis is not possible by Spin. Hence the model given by Steele and Andel abstracts from timing; as we have shown analysing OLSR with time variables reveals more shortcomings.

Fehnker et al. [8] describe a formal and rigorous model of the Ad hoc On-Demand Distance Vector (AODV) routing protocol in Uppaal; the model is derived from a precise process-algebraic specification that reflects a common and unambiguous interpretation of the RFC [19]. Their model is also a network of timed automata and they analyse network topologies up to 5 nodes. However, in their original analysis they abstract from time, which was added later on [10]. Although the two protocols AODV and OLSR behave differently, we use the same modelling techniques and experiments as for AODV, to make the comparison study of these two protocols feasible for our future work.

Kamali et al. [14] use refinement techniques for modelling and analysing wireless sensor-actor networks. They prove that failed actor links can be temporarily replaced by communication via the sensor infrastructure, given some assumptions. They use an Event-B formalisation based on theorem proving and their proofs are carried out in the RODIN tool platform. There is a strong similarity between the nature of the distributed OLSR protocol and the nature of distributed sensor-based recovery. However, the tools employed for analysis in the two frameworks are different in nature (model checking vs. theorem proving). Our decision to use Uppaal is based on the fact that it provides modelling means for time constraints and fully automatic reasoning. The treatment of time in Event-B is still incipient, involving a rather different perspective of treating variables as continuous functions of time.

6 Conclusions and Outlook

In this paper we have provided a formal analysis for the distributed and proactive routing protocol OLSR. Our analysis is performed using the model checker Uppaal. We have provided a Uppaal model which is in accordance with the OLSR standard. It models all core functionalities, including sophisticated timers. To validate our model we compared our model with examples found in the literature.

Using Uppaal we were able to find shortcomings of the protocol: in some cases, an optimal route for message delivery cannot be established and the recovery time in case of link breakage is huge. For both shortcomings we have sketched improvements that can easily be implemented. A more careful analysis for link breaks on critical paths is left for future work.

We see these results as the starting point for further research. First, our analysis is restricted to small networks (of 5 and 7 nodes), due to the nature of model checking. Wireless Mesh Networks draw their strength from employing potentially dozens (maybe hundreds) of nodes. Hence, we need to extend our analysis to larger networks. This can be achieved by working with statistical model checking, where simulation concepts are combined with model checking to establish the statistical evidence of satisfying hypotheses. While this does not guarantee a correct result w.r.t. the hypothesis, the probability of error can be made vanishingly small. Another approach suitable to deal with larger networks is that of theorem-proving, where, e.g. we can prove the required system properties as invariants for all systems (of all sizes) that verify certain assumptions.

Second, our model for the proactive, distributed OLSR can be generalised to distributed control. The latter is a concept with high relevance for systems where, e.g. self-repairing is important, as it can enable the independence of the system from central coordinators. Even maintaining proactively the optimal communication routes, as OLSR does, is instrumental in this. The applicability of distributed control to critical systems such as emergency response networks or smart electrical grids is very relevant, as these are complex systems, for which global solutions cannot be provided.

Acknowledgements. This research belongs to the Academy of Finland FResCo project (grant number 263925, FResCo: High-quality Measurement Infrastructure for Future Resilient Control Systems). NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
2. Chiyangwa, S., Kwiatkowska, M.: A timing analysis of AODV. In: Steffen, M., Zavattaro, G. (eds.) FMOODS 2005. LNCS, vol. 3535, pp. 306–321. Springer, Heidelberg (2005)

3. Clarke, E.M., Emerson, E.A., Sifakis, J.: Model checking: algorithmic verification and debugging. *Commun. ACM* **52**(11), 74–84 (2009)
4. Clausen, T., Jacquet, P.: Optimized link state routing protocol (OLSR). RFC 3626 (Experimental) (2003). <http://www.ietf.org/rfc/rfc3626>
5. David, A., Håkansson, J., Larsen, K.G., Pettersson, P.: Model checking timed automata with priorities using DBM subtraction. In: Asarin, E., Bouyer, P. (eds.) *FORMATS 2006*. LNCS, vol. 4202, pp. 128–142. Springer, Heidelberg (2006)
6. Emerson, E.A.: Temporal and Modal Logic. *Handbook of Theoretical Computer Science* (vol. B): Formal Models and Semantics. MIT, Cambridge (1995). pp. 995–1072
7. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Modelling and analysis of AODV in UPPAAL. In: 1st International Workshop on Rigorous Protocol Engineering, Vancouver, pp. 1–6 (2011)
8. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: Flanagan, C., König, B. (eds.) *TACAS 2012*. LNCS, vol. 7214, pp. 173–187. Springer, Heidelberg (2012)
9. van Glabbeek, R., Höfner, P., Portmann, M., Tan, W.L.: Sequence numbers do not guarantee loop freedom —AODV can yield routing loops—. In: *Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2013)*, pp. 91–100. ACM (2013)
10. Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: Brat, G., Rungta, N., Venet, A. (eds.) *NFM 2013*. LNCS, vol. 7871, pp. 322–336. Springer, Heidelberg (2013)
11. Holzmann, G.J.: The model checker spin. *IEEE Trans. Softw. Eng.* **23**(5), 279–295 (1997)
12. Jacquet, P., Mühlethaler, P., Clausen, T., Laouiti, A., Qayyum, A., Viennot, L.: Optimized link state routing protocol for ad hoc networks. In: *Multi Topic Conference, 2001, IEEE INMIC 2001*, pp. 62–68. IEEE (2001)
13. Kamali, M., Kamali, M., Petre, L.: Formally analyzing proactive, distributed routing. Technical report. 1125, TUCS - Turku Centre for Computer Science (2014)
14. Kamali, M., Laibinis, L., Petre, L., Sere, K.: Formal development of wireless sensor-actor networks. *Sci. Comput. Program.* **80**, Part A(0) **80**, 25–49 (2014)
15. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. *Int. J. Softw. Tools Technol. Transf. (STTT)* **1**(1), 134–152 (1997)
16. Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Compact data structures and state-space reduction for model-checking real-time systems. *Real-Time Syst.* **25**(2–3), 255–275 (2003)
17. Larsen, K.G., Pettersson, P., Yi, W.: Model-checking for real-time systems. In: *FCT*, pp. 62–88 (1995)
18. Miskovic, S., Knightly, E.W.: Routing primitives for wireless mesh networks: design, analysis and experiments. In: *Conference on Information Communications (INFOCOM 2010)*, pp. 2793–2801. IEEE (2010)
19. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental) (2003). <http://www.ietf.org/rfc/rfc3561>
20. Steele, M.F., Andel, T.R.: Modeling the optimized link-state routing protocol for verification. In: *SpringSim (TMS-DEVS)*, pp. 35:1–35:8. Society for Computer Simulation International (2012)

Paper II

Improved Recovery for Proactive, Distributed Routing

Mojgan Kamali and Luigia Petre

Originally published in: *proceedings of the 20th International Conference on Engineering of Complex Computer Systems (ICECCS 2015)*, IEEE proceeding, pp. 206 - 209, IEEE, 2015.

©2015 IEEE. Reprinted with permission from the publisher.

Improved Recovery for Proactive, Distributed Routing

Mojgan Kamali and Luigia Petre

Åbo Akademi University and Turku Centre for Computer Science (TUCS)

Turku, Finland

mojgan.kamali@abo.fi, luigia.petre@abo.fi

I. INTRODUCTION

Wireless Mesh Networks (WMNs) are self-organising ad-hoc networks that provide support for broadband communication without relying on a wired infrastructure. Such networks have gained popularity and are increasingly applied in a wide range of application areas, including telecommunication systems, emergency response networks, and intelligent transportation systems. In these and other instances, the communication is often truly distributed, without depending on any central entity (router) for coordination. As a consequence, automatic route-discovery, maintenance and repair has a significant impact on the reliability and performance of such networks.

An important feature of WMNs is that the network topology is changing. For instance, nodes might just fail in case of natural disasters; or nodes such as laptops and mobile phones can move within the network or even enter or leave a network in case of telecommunication systems. Hence, routes between source and destination nodes are usually not stable over time; one of the primary objectives of WMN routing protocols is to provide for such instability.

A routing protocol specifies how nodes communicate with each other in a network by disseminating information that enables them to select routes. In this way, nodes are able to send data packets to arbitrary (previously unknown) destinations in the network. Shortcomings in the routing protocol will lead to decreased performance and reliability of the entire network. Moreover, in case of topology changes (e.g., mobility, link failures), information has to constantly be updated to maintain the latest routing information within the network.

Performing an efficient and rapid recovery, i.e., reducing data loss and delays after a link breakage, is a significant feature and a challenging task. The efficiency largely depends on how fast the failure is detected and how fast corrective measures are applied by the routing protocol. A recovery mechanism highly relies on the nature of the routing protocol, i.e., proactive or reactive. In proactive protocols, routing toward nodes is always kept up-to-date, while reactive protocols build and update their routing information on demand.

The Optimised Link State Routing (OLSR) protocol [1], a proactive routing protocol, is identified as one of the standard ad-hoc routing protocol by the IETF MANET working group¹. By distributing control messages (HELLO and Topology Control (TC)) throughout the network, proactive protocols maintain a list of all destinations together with routes to them. When

the WMN faces a link breakage, remote nodes (that are not able to sense the breakage of the link on their own) would not be aware of this failure unless they receive control messages containing the relevant information. As a consequence, any formal analysis method for WMNs protocols should take link failure into account.

Formal analysis of specifications allows to systematically screen protocols for flaws as well as to present counterexamples to diagnose the flaws. Formal methods provide valuable tools for the design, evaluation, and verification of WMN routing protocols; they complement alternatives such as test-bed experiments and simulation. In this way, stronger and more general assurance about protocol behaviour and properties can be achieved. Model checking (e.g. [2]) is a powerful such formal approach used for validating key correctness properties in finite representations of a formal system model.

In this paper we extend our previous work [3] on formal analysis of the OLSR protocol using Uppaal. Uppaal [4], [5] is a well-established model checker for modelling, simulation and verification of real-time systems. It is based on the theory of timed automata [6], [7] and is used for systems that can be modelled as networks of (extended) timed automata. In [3] we have investigated the OLSR behaviour in case of route establishment, packet delivery, and optimal route finding on arbitrary small (static and dynamic) networks. Using model checking, in particular Uppaal, we have uncovered some malfunctioning in the behaviour of OLSR as well as pointed out a long recovery time in case of link failures. More details concerning OLSR can be found in its specification [1] and in [3]; a concrete instance of the OLSR behaviour for a network topology of 5 nodes can be found in [8]. The OLSR long recovery time can negatively affect the performance of the network. For a wide acceptance of WMNs, it is important that the recovery process does not imply unacceptable delays for networking applications (for instance in emergency response networks).

To address this, our approach in this paper is based on introducing a new type of message in OLSR, namely the *ERROR* message. We show how our modifications improve the recovery properties of OLSR when link failure occurs. Concretely, our contribution is twofold:

- We demonstrate that the recovery time in OLSR, due to link failures, can be reduced to half.
- We illustrate our findings with a simpler formal model, involving a reduced state space and time for verification.

More details about our proposed model and the results can

¹<http://datatracker.ietf.org/wg/manet/charter/>

be found in [9]. In Section II we outline a brief overview of our experimental results. A short discussion of our contribution as well as future work appear in Section III.

II. EXPERIMENTAL RESULTS

In this section we briefly describe the extension of our Uppaal model [3] for OLSR with ERROR messages. These messages are introduced to distribute the information about any link breakage in the network. In [3], we modelled OLSR as a parallel composition of identical processes describing the behaviour of nodes in the network. Each of these processes is in turn a parallel composition of two timed automata, *Queue* (input buffer of every node) and *OLSR* (main behaviour of the protocol) having their own local data structures.

According to OLSR's specification (RFC 3626), nodes broadcast HELLO and TC messages every $\text{time_between_hello} + \text{time_sending} = 2000$ milliseconds and $\text{time_between_tc} + \text{time_sending} = 5000$ milliseconds, respectively. These messages are used to update the information in routing tables. Information from these messages (HELLO and TC) is valid in the routing tables for $\text{NEIGHB_HOLD_TIME} = 6$ and $\text{TOP_HOLD_TIME} = 15$ seconds, respectively, after which, routing tables are reset. In our proposed model we first investigate the consequences of considering a shorter waiting interval. Namely, instead of refreshing routing tables after 6 and 15 seconds for the nodes that no HELLO or TC are received from, we consider a waiting interval equal to 4 seconds, in order to decrease the recovery time of OLSR. This means that if one node does not receive a HELLO message from its one-hop neighbour during 4 seconds, it resets its routing table for that node and starts broadcasting ERROR messages declaring breakage of the link between them. The full model describing this behaviour is available in [9].

We replay our experiments of [3] to verify and to compare the new model (applying our proposed solution) with our previous one (based on RFC [1]) on all network topologies up to 5 nodes. Our analysis of OLSR properties takes place in two different settings, namely in static network topologies and in dynamic network topologies where an arbitrary link fails. In the first setting, we focus on two properties to verify for our new model. These (safety) properties are as follows:

- (1) *route establishment* for all topologies up to 5 nodes;
- (2) *packet delivery* in all topologies up to 5 nodes.

In the second setting, we focus on the following properties:

- (3) the *shortened route discovery time in case of a link failure on a non-critical path*, i.e., we investigate the time needed to guarantee packet delivery.
- (4) the *shortened route discovery time in case of failing and adding a link on a critical path*, i.e., we investigate the time needed to guarantee packet delivery.

Importantly, we demonstrate how our new model shortens the long recovery time of OLSR discussed in our previous paper. Due to the proactive nature of OLSR, our previous Uppaal model was pretty complex, with a large data structure and multiple timers. As a consequence, state space explosion was a problem for our experiments. In our new model we have fewer

timers, leading to a manageable state space explosion and also a significantly reduced verification time. In the following we describe how our proposed solution helps to improve OLSR and its verification time.

For all Properties (Properties (1)–(4)), the originator is always the same node, denoted *OIP1*, and the destination is always the same node, denoted *DIP1*. For the experiments we used the following setup: 3.2 GHz Intel Core i5, with 8 GB memory, running the Mac OS X Yosemite 10.10.3 operating system. For all experiments we use Uppaal 4.1.19.

A. Static Network Topologies

For evaluating and verifying our proposed model in static network topologies, we define an automaton called *Packet_injecting*. This automaton injects a data packet to node *OIP1* to reach the destination node *DIP1* as shown in Fig. 1. This automaton has a local clock *clk* to guarantee that the packet is injected to node *OIP1* via the channel *packet[OIP1][DIP1]* at time $3 * (\text{time_between_tc} + \text{time_sending})$.

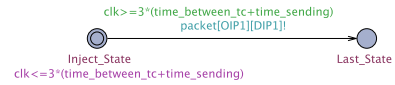


Fig. 1: *Packet_injecting* automaton modeling packet injection in static networks.

Property (1), route establishment property, states that if the packet has been injected (*Packet_injecting* is in location *Last_State*), and all messages have been processed (*emptybuffers()*), then OLSR has established a route between *OIP1* and *DIP1*. This safety property is expressed as following using the Uppaal syntax:

```
A[] ((Packet_injecting.Last_State && emptybuffers())
      imply node(OIP1).rt[DIP1].nhopip! = 0) (1)
```

Here, $A[]\phi$ is satisfied iff ϕ holds on all states along all paths. The variable *node(OIP1).rt* denotes the routing table of the originator node *OIP1* and *node(OIP1).rt[DIP1].nhopip* represents the next hop for the destination *DIP1*: if the next hop is not 0, then a route is established.

Property (2), packet delivery property, states that if the packet is injected to the system (*Packet_injecting* automaton is in location *Last_State*), then it is delivered at the destination *DIP1*. This property is expressed as

```
A[] ((Packet_injecting.Last_State && emptybuffers())
      imply node(DIP1).delivered! = 0) (2)
```

Here, *node(DIP1).delivered* shows whether the injected data packet is received by the destination *DIP1*.

Property (2) is stronger than Property (1) in the sense that the established route must be correct and used. In reality, an originator node might have a routing table entry for the destination node in the network, indicating that it can send a packet to the next node along the path to the destination node; however, the next node itself could have no information about the destination node, so all packets stemming from the originator node for the destination would be lost. In other words, the next node along the path to the destination node might have reset its routing table for the destination since its local clock for the destination node exceeds 4 seconds.

The first two experiments are performed for all topologies up to five nodes, up to isomorphism and renaming.

Results: For analysing and verifying our OLSR model, we evaluate Properties (1) and (2) in all network topologies up to 5 nodes. Property (1) is satisfied for all these networks stating that when the `Packet_injecting` is in the `Last_State` location, the originator node `OIP1` has established a route to the destination node `DIP1`. This property confirms the propagation of HELLO and TC messages, correct functionality of updating routing tables at our proposed timing (4 seconds). Hence, node `OIP1` has the required information about the node `DIP1` to send the data packet.

As mentioned before, Property (2) is stronger than Property (1). It states that all the nodes have the information about all other nodes in the network, to deliver their data packets. Property (2) is also satisfied for all topologies up to size 5: all nodes have the information about all other nodes in the network in their routing tables; therefore, they are ready to deliver data packets to the arbitrary destination node `DIP1`.

B. Dynamic Network Topologies

Removing a Link: For evaluating and verifying our proposed model in dynamic network topologies, we define another automaton, called `Link_dropping`, which drops the link between the two nodes `OLSR1` and `OLSR2` and is depicted in Fig. 2. `Link_dropping` provides a local clock `clk` to guarantee link failure at time $3 * (\text{time_between_tc} + \text{time_sending})$ when all nodes have information about all other nodes in the network (all routing tables have been updated for all nodes). Upon link breakage, there is no connectivity between these two nodes; however each of them has the information about the other one. As soon as a link break is identified (local clock `t_reset_rt` exceeds 4 seconds), error messages should be sent to inform other nodes about the link breakage and to correct the information in the routing tables as soon as possible. In this way, we are able to analyse how fast OLSR recovers from topology changes broadcasting `ERROR` messages.

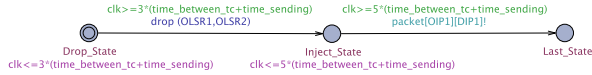


Fig. 2: `Link_dropping` automaton modelling packet injection in dynamic networks without critical link.

The data packet is injected to the network at time $5 * (\text{time_between_tc} + \text{time_sending})$ to check whether or not the packet is delivered at the destination. We consider the property which indicates whether or not the injected packet is delivered at the destination if one link is removed. In Uppaal syntax this safety property can be expressed as:

$$A[] ((\text{Link_dropping.Last_State} \ \&\& \ \text{emptybuffers}()) \implies \text{node}(\text{DIP1}).\text{delivered}! = 0) \quad (3)$$

After the link is removed (the topology has been changed) and the data packet has been injected, the automaton `Link_dropping` is in location `Last_State`. At that time, there are no messages in the buffers (all messages have been handled by all the nodes) if and only if function `emptybuffers` returns `true`: then we investigate if the packet has been delivered, i.e., $(\text{node}(\text{DIP1}).\text{delivered}! = 0)$.

Results: Property (3) is satisfied for those topologies up to 5 nodes where the dropped link is not critical². This experiment indicates that the recovery in these topologies (5 nodes networks) takes around 10 seconds (the difference between 15000–25000 milliseconds), i.e., half the recovery time in our previous work [3]. We should mention here that in [3], we verified that the recovery time of OLSR takes 20 seconds based on RFC [1].

Adding a Link: To verify packet delivery property for networks where the dropped link is a critical link, we define another automaton called `Link_adding` as depicted in Fig. 3. This automaton adds an arbitrary link between the two nodes `OLSR1` and `OLSR3` after the link breakage to keep the connectivity in the network. This link is added at time $4 * (\text{time_between_tc} + \text{time_sending})$ (20000 milliseconds) to the network. Then, the packet is injected at time $5 * (\text{time_between_tc} + \text{time_sending})$ (25000 milliseconds) to verify if the added link provides the connectivity between nodes to deliver the packet. This safety property is represented as:

$$A[] ((\text{Link_adding.Final_State} \ \&\& \ \text{emptybuffers}()) \implies \text{node}(\text{DIP1}).\text{delivered}! = 0) \quad (4)$$

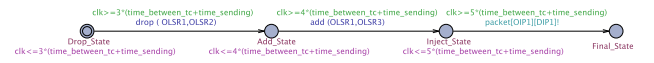


Fig. 3: `Link_adding` automaton modelling packet injection in dynamic networks with critical link.

Property (4) states that if the automaton `Link_adding` is in the `Final_State` location and there is no message in buffers, then the packet must be delivered at the destination. If this property is satisfied, it shows that by adding an arbitrary link the network can be recovered from the failure even in networks with critical links. Upon a link is added to the network, the neighbour nodes would immediately be notified about the new link by receiving HELLO messages so that they can establish new routes.

Results: As for Property (3), Property (4) is also satisfied for all topologies up to 5 nodes even in networks where the dropped link is a critical link. This experiment shows that the recovery time of such networks (5 nodes networks) also takes about 10 seconds.

Introducing `ERROR` messages for OLSR and modifying the `NEIGHB_HOLD_TIME` value helps to shorten the long recovery time of OLSR: the number of control messages with incorrect information is reduced since routing tables are updated when receiving `ERROR` messages. As a consequence, nodes will not broadcast control messages with outdated information (when a node receives an `ERROR` message containing the information about any link breakage in the network, the receiving node updates its routing tables based on the information in the `ERROR` message, so it will broadcast fresh information in its next broadcasting).

C. Comparison

In the described experiments, we have verified the properties of our alternative solution, especially with respect to the

²In our model, a critical link is a link after whose breakage there is no other link from that node to the other nodes along the path to the destination to be substituted with the broken one.

long recovery time of OLSR: we have found that our proposal helps to decrease this interval to about 10 seconds. In this section, we compare our new model (with ERROR messages) with our previous model (based on RFC 3626).

In spite of introducing new messages, our new model uses less time and memory for the verification. This is due to the fact that we minimised the complexity of our model by reducing the number of timers and instead defining the ERROR new type of messages, to propagate the information about link breakage. Remember that in model [3], we had one more array of clocks which was used to delete the information if no TC message is received during 15 seconds thus caused more time for the verification and increased the system complexity. Hence, in addition to decreasing the recovery time of OLSR, we also reduce the verification time and the number of explored states, thus contributing to overcome the state space explosion problem. Reducing the complexity of the system makes it feasible for us to verify the system for larger networks also. More details and also figures depicting the differences between both models can be found in [9].

III. DISCUSSION

While improving the performance of network protocols is not a new research topic, attempts to formal analysis and improvement of routing protocols are still rather new and remain a challenging task. In particular, the formal analysis and verification of MANETs and ad-hoc routing protocols received a lot of attention from the formal methods community [3], [10]–[12]. For example, Fehnker et al. [13] describe a formal and rigorous model of the AODV routing protocol in Uppaal; this is derived from a precise process-algebraic model that reflects a common and unambiguous interpretation of the RFC [14]. They were able to automatically locate problematic and undesirable behaviour of AODV and to propose and verify some modifications to improve the performance of such protocol.

Our work has been motivated by investigating the long recovery time of a distributed and proactive routing protocol named OLSR [3]. In this paper we modelled and verified our proposed modification with respect to alleviating the recovery time of OLSR by introducing new type of messages known as ERROR messages. Our analysis is performed using the model checker Uppaal.

Using Uppaal we were able to demonstrate the shortening of an otherwise long recovery time for the OLSR protocol; moreover, we were also able to reduce the time used for the verification. A smaller routing table updating time (4 seconds) and propagating ERROR messages lead to a faster detection of broken links. In comparison to simulation studies, our analysis based on rigorous model checking enabled us to give a good overall view of the performance and behaviour in any situation for networks up to 5 nodes. Our analysis was focused on small networks (because of the nature of model checking): we need next to extend our analysis to larger networks. This can be achieved, for instance, by applying statistical model checking on the current model and/or theorem proving techniques by defining a new model.

We have also analysed the system (model) for link failure on critical paths where an arbitrary link was added to the

network. Our formal modelling and analysing was extended to recover broken paths and so to address the situation where the failed link (Section II-B) is a critical link. This is important, because of the self-healing and self-organising nature of WMNs, essential for rescue applications and safety critical systems. In addition, the model for OLSR is general enough to allow for the study of more complicated scenarios, in particular mobility. We plan to study a number of mobility models to investigate the behaviour of OLSR and other routing protocols.

ACKNOWLEDGMENT

This research belongs to the Academy of Finland FResCo project (grant number 263925, FResCo: High-quality Measurement Infrastructure for Future Resilient Control Systems).

REFERENCES

- [1] T. Clausen and P. Jacquet, “Optimized link state routing protocol (OLSR),” RFC 3626 (Experimental), Internet Engineering Task Force, 2003. [Online]. Available: <http://www.ietf.org/rfc/rfc3626>
- [2] E. M. Clarke, E. A. Emerson, and J. Sifakis, “Model checking: Algorithmic verification and debugging,” *Commun. ACM*, vol. 52, no. 11, pp. 74–84, 2009.
- [3] M. Kamali, P. Höfner, M. Kamali, and L. Petre, “Formal analysis of proactive, distributed routing,” in *13th International Conference on Software Engineering and Formal Methods (SEFM 2015)*, vol. 9276. Springer, 2015, pp. 175–189.
- [4] G. Behrmann, A. David, and K. G. Larsen, “A tutorial on Uppaal,” in *International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004. Revised Lectures*. Springer Verlag, 2004, pp. 200–236.
- [5] K. G. Larsen, P. Pettersson, and W. Yi, “Uppaal in a nutshell,” *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, 1997.
- [6] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2006.
- [7] R. Alur and D. L. Dill, “Automata for modeling real-time systems,” in *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*. Springer-Verlag New York, Inc., 1990, pp. 322–335.
- [8] M. Kamali, M. Kamali, and L. Petre, “Formally analyzing proactive, distributed routing,” TUCS – Turku Centre for Computer Science, Tech. Rep. 1125, 2014.
- [9] M. Kamali and L. Petre, “Improved recovery for proactive, distributed routing,” TUCS – Turku Centre for Computer Science, Tech. Rep. 1143, 2015.
- [10] A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan, “Modelling and analysis of AODV in UPPAAL,” in *1st International Workshop on Rigorous Protocol Engineering*, 2011, pp. 1–6.
- [11] D. Benetti, M. Merro, and L. Vigan, “Model checking ad hoc network routing protocols: Aran vs. endaira,” in *8th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2010)*. IEEE Computer Society, 2010, pp. 191–202.
- [12] S. Chiyangwa and M. Z. Kwiatkowska, “A timing analysis of AODV,” in *FMOODS*, ser. Lecture Notes in Computer Science, vol. 3535. Springer, 2005, pp. 306–321.
- [13] A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan, “Automated analysis of AODV using UPPAAL,” in *18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012)*. Springer, 2012, pp. 173–187.
- [14] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing,” RFC 3561 (Experimental), Internet Engineering Task Force, 2003. [Online]. Available: <http://www.ietf.org/rfc/rfc3561>

Paper III

Modelling Link State Routing in Event-B

Mojgan Kamali and Luigia Petre

Originally published in: *proceedings of the 21st International Conference on Engineering of Complex Computer Systems (ICECCS 2016)*, IEEE proceeding, pp. 207-210, IEEE, 2016.

©2016 IEEE. Reprinted with permission from the publisher.

Modelling Link State Routing in Event-B

Mojgan Kamali and Luigia Petre

Åbo Akademi University

Turku, Finland

mojgan.kamali@abo.fi, luigia.petre@abo.fi

Abstract—In this paper we present a stepwise formal development of the Optimised Link State Routing (OLSR) protocol in Event-B. OLSR is a proactive routing protocol which finds routes for different destinations in advance by exchanging control messages through the network. As a consequence, whenever a data packet is injected into the network can be delivered to a certain destination immediately. To achieve this, routing tables in OLSR are continuously updated, by following a rather complicated algorithm. By modelling OLSR in Event-B, we address the scalability problem of our previous work [1], and structure the OLSR complexity in five distinct abstraction layers. These layers are manageable to understand and to verify and are linked to each other by refinement. As Event-B is supported by a theorem proving platform (Rodin), we model and prove functional properties of OLSR in an automated and interactive manner, at a highly general level. Our approach can serve as a proof-of-concept to be adapted to modelling and verifying of the other routing protocols for large-scale networks.

I. INTRODUCTION

Wireless technologies are on the rise, ranging from laptops and smart phones that make work and connections easier, to sensor networks that produce and manipulate large amounts of data. In this study, we focus on contemporary wireless networks, in particular Wireless Mesh Networks (WMNs). WMNs are self-healing and self-organising wireless technologies supporting broadband communication without requiring any wired infrastructure. These networks have gained popularity and are employed in a wide range of application areas such as emergency response networks, communication systems, video surveillance, etc. Due to these critical applications, correct behaviour and functioning of such networks should be guaranteed. One of the key factors determining the performance and reliability of WMNs is their routing protocols.

The fundamental role of (WMN) routing protocols is to provide routes between nodes for communication. These protocols disseminate information through the network in order to select such routes and thus enable nodes to send data packets to arbitrary destinations in the network. Routing protocols for WMNs are either proactive or reactive. Proactive protocols attempt to select routes in advance, by exchanging control messages about all the other nodes of the network. Consequently, an injected data packet can be delivered to the destination immediately. Examples of such protocols are Optimised Link State Routing (OLSR) protocol, Better Approach To Mobile Ad hoc Networking (BATMAN) routing protocols, etc. Reactive protocols search for routes to destination nodes on demand, whenever a data packet is injected into the network. Examples of reactive protocols are Ad hoc On-Demand

Distance Vector (AODV) protocol, Dynamic Source Routing (DSR) protocol, etc.

In this paper we focus on the OLSR protocol [2]. We formally model this protocol through incremental stepwise refinements in Event-B [3]. We use Event-B as a formal modelling approach which provides automatic tool support for modelling and proving various properties.

OLSR is a rather complicated protocol. This is mainly due to its proactive nature, requiring it to keep up-to-date information about routes to all destinations from any node. Assumed to work in the WMN setup, this implies that all routes are checked periodically. Indeed, the OLSR specification [2] prescribes certain routines to be performed every 2 seconds, and some other every 5 seconds. There are two kinds of control messages exchanged by nodes for updating the complex structure of the routing tables.

By stepwise modelling OLSR in Event-B we gain a deep understanding of the OLSR mechanisms at five different layers of abstraction. The first two layers simply model a routing protocol, first more abstractly and then more concretely. The following two layers add the infrastructure necessary for modelling the proactive behaviour, again in a more abstract manner first and somewhat more concretely in layer four. In the final layer we add the defining characteristic of OLSR, of selecting only specific nodes to broadcast control messages, so as not to strangle the network traffic. These layers are related by refinement: this means that the properties and the behaviour of any model are kept in all its subsequent refinements. Hence, we can prove certain properties in a more abstract layer (when the property is simpler to prove) and then develop the models in the more concrete layers so that they do not break those proven properties.

TABLE I. LEVELS OF ABSTRACTION

Levels of abstraction	Description
$M0$	Abstract general communication protocol considering simple communication between only two nodes (without forwarding)
$M1$	Concrete general communication protocol considering general communication between different nodes (with forwarding)
$M2$	Abstract OLSR communication protocol considering the exchange of control messages and updating routing tables
$M3$	More detailed OLSR communication protocol considering the process of only fresh (new) messages using sequence numbers in the messages and routing tables
$M4$	Detailed OLSR communication protocol considering that only some special nodes broadcast control messages to decrease the network traffic

Our contribution is thus threefold:

- We model OLSR protocol at different levels of abstraction (TABLE I) and prove that the more concrete models refine the more abstract ones. By this, we demonstrate the management of complexity in an elegant manner to offer our solution as a proof-of-concept for other routing protocols to be modelled and verified.
- We address the scalability problem in our previous work [1] where we used model checking technique to model OLSR formally. The main concern with respect to [1] was the number of nodes in the network since model checking technique suffers from the state space explosion problem. This paper is a significant case study in the sense that it deals with developing and verifying the real routing protocols in large-scale (realistic) networks.
- As a consequence of the specific properties we prove, we discover that:
 - OLSR does not find optimal routes to all the destinations,
 - it indeed discovers routes to all the destinations,
 - it delivers data packets on these discovered routes (considering networks of arbitrary size).

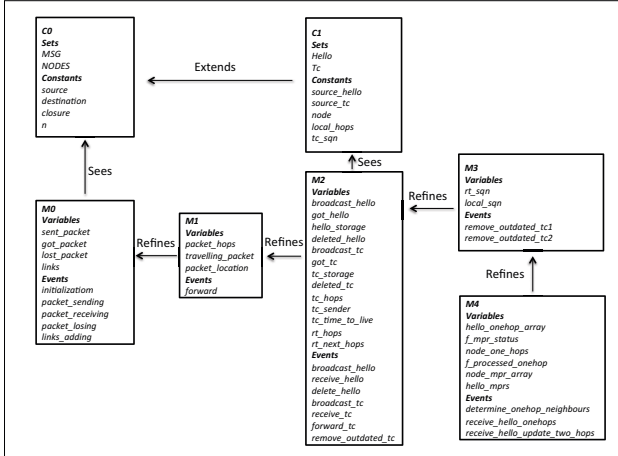


Fig. 1. Overview of model development

More details about our model and the results can be found in [4]. We present a brief overview of our OLSR formal development in Section II. Verification results and the impact of our paper are described in Section III. We draw some conclusions as well as some future work in Section IV.

II. EVENT-B MODEL OF OLSR

In this section, we express the OLSR protocol development at five different layers of abstraction, overviewed below. We summarise our development of the OLSR model in Fig. 1, where we illustrate the five abstraction layers containing the corresponding five machines as well as two contexts.

Initial Model (M0): Basic routing protocol behaviour, i.e., sending, receiving, and losing data packets, is specified, together with an abstraction of proactive routing behaviour.

Refinement 1 (M1): A storing and forwarding architecture for data packets from a source node to a destination node is modelled in this step.

Refinement 2 (M2): The basic behaviour of route discovery is introduced in this refinement. This level describes the essential OLSR behaviour for sending and receiving control messages based on which each node updates its routing table.

Refinement 3 (M3): More detailed information about the route discovery protocol and how to process new control messages are provided in this level of refinement.

Refinement 4 (M4): Selection of MPR nodes and how they help to decrease the traffic in the network are introduced in this step.

The refinement relation between machines is emphasised in Fig. 1 and we illustrate the *new* variables and events in all the machines, together with constants and carrier sets in the contexts. In the following sections we describe some of these entities. All the details are only described in our technical report [4] due to lack of space. The reader can also consult our models in Event-B or in pdf format at http://users.abo.fi/mokamali/ICECCS_2016.

A. The Context C0 and Initial Model M0

In context C0 we define two carrier sets *NODES* and *MSG* and four constants *source*, *destination*, *closure*, and *n*. The carrier set *NODES* models the nodes in the network; this set is finite and non-empty (modelled by *axm1* and *axm2*, respectively). The number of elements in *NODES* is equal to *n* (*axm3*) which is larger than 1 (*axm4*). In axioms *axm5* and *axm6*, we define the carrier set *MSG* as a finite and non-empty set, modelling the set of all (user) data packets. In axioms *axm7* and *axm8*, we define the type of the constants *source* and *destination*, modeling the source and, respectively, the destination of all data packets; these are total functions mapping *MSG* to *NODES*. The constant *closure* models the transitive closure of binary relations between nodes (*NODES*) in axioms *axm9*–*axm12*. The sets, constants and axioms belong to the context C0, seen by our initial model M0.

```

SETS
  MSG, NODES

CONSTANTS
  source, destination, closure, n

AXIOMS
  axm1 : finite(NODES)
  axm2 : NODES ≠ ∅
  axm3 : card(NODES) = n
  axm4 : n > 1
  axm5 : finite(MSG)
  axm6 : MSG ≠ ∅
  axm7 : source ∈ MSG → NODES
  axm8 : destination ∈ MSG → NODES
  axm9 : closure ∈ (NODES ↔ NODES) →
    (NODES ↔ NODES)
  axm10 : ∀ r. r ⊆ closure(r)
  axm11 : ∀ r. closure(r); r ⊆ closure(r)
  axm12 : ∀ r, s. r ⊆ s ∧ s; r ⊆ s ⇒ closure(r) ⊆ s

```

In M0, we model an abstract version of the OLSR protocol. We have four variables, namely *sent_packet*, *lost_packet*, *got_packet* and *links*. Variable *sent_packet* is a subset of *MSG* (*inv1*) modelling the packets actually sent through the network (injected). Lost packets and received packets in the network are modelled by variables *lost_packet* and *got_packet*, respectively. The set of lost packets (*lost_packet*) and received packets (*got_packet*) are subsets of the set of all injected packets (*sent_packet*); this is a safety property modelled by invariants *inv2* and *inv3*. A data packet cannot be received and lost at

the same time as modelled in *inv4*. Variable *links* models the current links in the network (*inv5*). No node is connected to itself, as modelled by invariant *inv6*.

INVARIANTS

```

inv1 :  $\text{sent\_packet} \subseteq \text{MSG}$ 
inv2 :  $\text{lost\_packet} \subseteq \text{sent\_packet}$ 
inv3 :  $\text{got\_packet} \subseteq \text{sent\_packet}$ 
inv4 :  $\text{got\_packet} \cap \text{lost\_packet} = \emptyset$ 
inv5 :  $\text{links} \in \text{NODES} \leftrightarrow \text{NODES}$ 
inv6 :  $(\text{NODES} \triangleleft \text{id}) \cap \text{links} = \emptyset$ 

```

There are four simple events in our abstract model in addition to the initialisation event where all variables get value \emptyset . The event *packet_sending* models the sending of a data packet *msg* not yet injected from a source node *s* to a destination node *d* (*grd1*–*grd3*). The main guard of this event ensures that there is a path from *s* to *d* (*grd4*). If these conditions hold, then *msg* can be sent (injected in the network to eventually make its way to *d*).

```

Event packet_sending  $\hat{=}$ 
  any
  where
    s, d, msg
    grd1 :  $\text{msg} \in \text{MSG} \wedge \text{msg} \notin \text{sent\_packet}$ 
    grd2 :  $\text{source}(\text{msg}) = s \wedge \text{destination}(\text{msg}) = d$ 
    grd3 :  $s \neq d$ 
    grd4 :  $s \mapsto d \in \text{closure}(\text{links})$ 
  then
    act1 :  $\text{sent\_packet} := \text{sent\_packet} \cup \{\text{msg}\}$ 
end

```

Event *packet_receiving* models the successful receiving of the data packet *msg* by a destination node. The guard of this event (*grd1*) models that *msg* has not been received or lost yet.

```

Event packet_receiving  $\hat{=}$ 
  any
  where
    msg
    grd1 :  $\text{msg} \in \text{sent\_packet} \setminus (\text{got\_packet} \cup \text{lost\_packet})$ 
  then
    act1 :  $\text{got\_packet} := \text{got\_packet} \cup \{\text{msg}\}$ 
end

```

Event *packet_losing* models loss of data packets. The guard of this event (*grd1*) models that *msg* has not been received or lost yet.

```

Event packet_losing  $\hat{=}$ 
  any
  where
    msg
    grd1 :  $\text{msg} \in \text{sent\_packet} \setminus (\text{got\_packet} \cup \text{lost\_packet})$ 
  then
    act1 :  $\text{lost\_packet} := \text{lost\_packet} \cup \{\text{msg}\}$ 
end

```

Event *links_adding* models that some arbitrary links not yet in the network (*grd1*) may come up; these new links are added to the set *links*.

```

Event links_adding  $\hat{=}$ 
  any
  where
    s, d
    grd1 :  $s \mapsto d \notin \text{links}$ 
    grd2 :  $s \neq d$ 
  then
    act1 :  $\text{links} := \text{links} \cup \{s \mapsto d\}$ 
end

```

We note here that the first three events of *M0*, *packet_sending*, *packet_receiving* and *packet_losing* are events common to any routing protocol. The specific proactive feature that routes to destinations are continuously updated in the

routing table, i.e., the valid links are continuously updated, is modelled in *M0* abstractly. This has been modelled in our fourth event *links_adding*. Later in the refinement chain, adding links in the network will be replaced by updating routing tables based on the information received from HELLO and TC messages. At this abstract level, however, we only have data packets as messages. Control messages are introduced in *M2*. We also note that receiving data packets happens abstractly, with having no intermediate nodes in between the source and the destination of a data packet. A sent data packet can be either received or lost in a non-deterministic manner. We add intermediate nodes in between sources and destinations in *M1*.

B. First Refinement M1: Storing and Forwarding Architecture

In the initial model, data packets are received in an atomic abstract step from a source node to a destination node. This is of course not the case in a real protocol. Data packets are transferred using multi-hop communication and they are forwarded hop by hop from a source node *s* to destination node *d*. Hence, in this refinement step, we model the storing and forwarding architecture of data packets when not all the nodes are directly connected and the data packet has to be forwarded by several intermediate nodes before being delivered at the destination node. For this, we define three new variables and one new event.

C. Second Refinement M2: Route Discovery Protocol

The route discovery protocol is the most important and complicated refinement step of this model. In this level of refinement, we investigate whether or not nodes can find optimal routes to different destination nodes. We add OLSR control messages (HELLO and TC) and model the routing tables of every node. In this step, we replace the centralised functioning of the routing protocol in models *M0* and *M1* with a distributed functioning by data refinement.

D. Third Refinement M3: Role of Sequence Numbers

In our third refinement we model and take into account sequence numbers of TC messages, to avoid processing TC messages with old information. It means that if the sequence number of the received TC message is bigger than the sequence number of the TC message originator in the corresponding routing table of the receiving node, then the TC message is processed and may be forwarded to the other nodes. Otherwise, the TC message is removed from the network.

E. Fourth Refinements M4: MPRs Selection

In our fourth refinement, we restrict the broadcasting of TC messages to only specific nodes, the MPRs. Only these nodes are able to broadcast and forward TC messages through the network which helps to decrease the traffic in the network. Both refinements optimise the protocol model significantly. We describe them in detail only in our technical report [4], due to lack of space. Our full models are also available at http://users.abo.fi/mokamali/ICECCS_2016, both as Event-B models and in pdf format.

III. VERIFICATION AND IMPACT

Figuring out from which level of abstraction to start modelling and what details to add at every step of the refinement is a challenging task. We model our system in order to address modelling/proving complexity and to preserve reusability of

TABLE II. PROOF STATISTICS

Model	Number of Proof Obligations	Automatically Discharged	Interactively Discharged
<i>C0</i>	4	4	0
<i>M0</i>	13	13	0
<i>M1</i>	51	47	4
<i>C1</i>	1	1	0
<i>M2</i>	102	95	7
<i>M3</i>	17	16	1
<i>M4</i>	34	23	11
Total	222	199	23

models. In order to check if our models satisfy their correctness properties we need to prove that the invariants are preserved. In our case, correctness of packet delivery after injecting a data packet to the system, optimal route finding, and route discovery are investigated. In order to prove these, we used the Rodin platform tool to generate the proof obligations for all the models. The summary of proof statistics is displayed in Table II consisting of the number of proof obligations generated by Rodin platform, number of proof obligations discharged automatically and number of proof obligations discharged interactively.

In addition to proving certain properties that hold for the OLSR protocol, our main contribution consists in the unpacking of OLSR complexity into five abstraction layers, as illustrated in Fig. 1. The first two layers describe the abstract behaviour of a routing protocol; the following two add the needed infrastructure to model proactive routing behaviour; and the last layer only introduces the specifics of OLSR, that of MPR-based working. Hence, our modelling is highly reusable: routing protocols can be developed based on the first two abstraction layers, even reactive routing protocols; other proactive routing protocols can be developed based on abstraction layers three and four, for instance the BATMAN routing protocol. Hence, we have presented a case study that emphasises the power of refinement-based methods as well as their reusability and adaptability. Reusing our models *M0* – *M3* in modelling other routing protocols is left for future research.

IV. CONCLUSIONS

Creating a model in Event-B for analysing the OLSR protocol was originally motivated by our previous work on creating an Uppaal model [1] for this protocol. While we could experiment with Uppaal on various properties, report some problems of the protocol, sketch some modifications [5] to fix these problems and improve the performance of OLSR, the main drawback of the Uppaal model was its limitation to 5-node topologies. We needed to understand whether OLSR works as it was intended for arbitrary topologies and thus, we created the Event-B model in this paper. This model also has some limitations, such as abstracting away from the timing behaviour and not including the deletion (failure) of links. We believe that the former drawback simply needs a more involved modelling approach, as we explain below. A deeper modelling approach is also needed for addressing the latter drawback, with the additional comment that in Uppaal we have studied what happens if a particular link in a particular 5-node topology fails.

In this paper, we have modelled the behaviour of OLSR and verified three main properties with respect to:

- finding routes to all destinations,
- delivering data packets along these routes,
- showing that OLSR does not find optimal routes for all the destinations.

These properties hold true also for our previous Uppaal model. However, the main advantages of our Event-B model are generality and reusability. Our proved management of OLSR complexity is a central contribution. The formal development of the protocol is carried out in several layers: routing protocol, proactive behaviour protocol, and OLSR. These layers are reusable and adaptable, as argued in Section III. The last layer elegantly shows how OLSR is a refinement of proactive routing, delivering a more efficient algorithm: only the selected MPR nodes are enabled to send and forward control messages, so as not to flood the network.

We plan to continue the research reported in this paper in several directions. First, we are working on the finishing touches of a companion paper where we compare the Uppaal model with the Event-B model for OLSR; in that paper we also discuss the relative advantages of using Uppaal and Event-B for analysing protocols. Second, there is timing behaviour in OLSR [2] that we abstracted away in this paper. By including it though, we would be able to reason about timing properties of OLSR, hence we plan to employ an approach for time modelling in Event-B, such as Hybrid Event-B [6] or [7] for instance. This would imply that all variables except clocks are functions of time, so a slight change of perspective is needed here. Third, there is a remarkable resemblance between the basic behaviour of data packets and the other control messages in the network: they are all sent, received, locally stored. Hence, we plan to investigate a theory of messages in connection with routing protocols in Event-B, much in the spirit of other theories [8] already introduced in the Rodin platform. This would increase the reusability of both the proposed models and proofs. Finally, showing how to reuse the general models, introduced in this paper for developing other routing protocols would clearly demonstrate the advantages of using Event-B and the refinement approach.

REFERENCES

- [1] M. Kamali, P. Höfner, M. Kamali, and L. Petre, “Formal analysis of proactive, distributed routing,” in *13th International Conference on Software Engineering and Formal Methods (SEFM 2015)*, vol. 9276. Springer, 2015, pp. 175–189.
- [2] T. Clausen and P. Jacquet, “Optimized link state routing protocol (OLSR),” RFC 3626 (Experimental), Internet Engineering Task Force, 2003. [Online]. Available: <http://www.ietf.org/rfc/rfc3626>
- [3] J. R. Abrial, *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.
- [4] M. Kamali and L. Petre, “Modelling link state routing in Event-B,” TUCS – Turku Centre for Computer Science, Tech. Rep., 2016.
- [5] M. Kamali and L. Petre, “Improved recovery for proactive, distributed routing,” in *20th International Conference on Engineering of Complex Computer Systems (ICECCS 2015)*. IEEE, 2015, pp. 178–181.
- [6] R. Banach, M. Butler, S. Qin, N. Verma, and H. Zhu, “Core hybrid event-b i: Single hybrid event-b machines,” *Science of Computer Programming*, vol. 105, pp. 92 – 123, 2015.
- [7] J. R. Abrial, W. Su, and H. Zhu, “Formalizing hybrid systems with event-b,” in *Abstract State Machines, Alloy, B, VDM, and Z*, ser. Lecture Notes in Computer Science, J. Derrick, J. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, Eds. Springer, 2012, vol. 7316, pp. 178–193.
- [8] M. Butler and I. Maamria, “Practical theory extension in event-b,” in *Theories of Programming and Formal Methods*, ser. Lecture Notes in Computer Science, Z. Liu, J. Woodcock, and H. Zhu, Eds. Springer, 2013, vol. 8051, pp. 67–81.

Paper IV

Uppaal vs Event-B for Modelling Optimised Link State Routing

Mojgan Kamali and Luigia Petre

Originally published in: *proceedings of the 11th International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS 2017)*, Lecture Notes in Computer Science, pp. 189-203, Springer, 2017.

©2017 Springer. Reprinted with permission from the publisher.

Uppaal vs Event-B for Modelling Optimised Link State Routing

Mojgan Kamali^(*) and Luigia Petre

Åbo Akademi University, Turku, Finland
{mojgan.kamali,lpetre}@abo.fi

Abstract. In this paper we compare models developed in two formal frameworks, Uppaal and Event-B, for the Optimised Link State Routing (OLSR) protocol. OLSR is one of the proactive routing protocols used in Mobile Ad-hoc Networks (MANETs) and Wireless Mesh Networks (WMNs). We also describe different aspects of the Uppaal and Event-B formalisms. This leads to a more general comparison of both formalisms, considering the following criteria: their specification languages, their update of variables mechanism, their modularity methods, their verification strategies, their scalability potentials and their real-time modelling capabilities. Based on it, we provide several guidelines for when to use Uppaal or Event-B for formal modelling and analysis.

1 Introduction

Continuous connectivity is a defining feature of our current working routines as well as of our free-time ones. We expect to be able to access information at all times as well as be able to communicate to various entities at all times. Technically, this is ensured with myriads of interconnected networks that offer us coverage and route all our requests for information and communication in certain ways. Hence, routing is a fundamental stone of our lifestyles and as such, presents enormous interest for study. Routing is obviously not a new concept for the era of continuous connectivity; it has been around since the first networks were developed some decades ago. Along with network evolution, routing has however evolved as well, with numerous algorithms in use today.

Routing protocols are divided into two main categories: proactive and reactive. Proactive protocols select routes in advance, by having network nodes exchanging (control) messages about all the other network nodes. Consequently, an injected data packet can be delivered to the destination immediately. Examples of such protocols are Optimised Link State Routing (OLSR) protocol [10], Better Approach To Mobile Ad hoc Networking (BATMAN) routing protocol [22], etc. Reactive protocols search for routes to destination nodes on demand, whenever a data packet is injected into the network. Examples of reactive protocols are Ad hoc On-Demand Distance Vector (AODV) protocol [23], Dynamic Source Routing (DSR) protocol [14], etc.

In this paper we compare two models for the OLSR proactive protocol. This protocol is used for routing in Wireless Mesh Networks (WMNs). WMNs are

self-healing and self-organising wireless technologies supporting broadband communication without requiring any wired infrastructure. They are employed in a wide range of application areas such as emergency response networks, communication systems, video surveillance, etc. A central feature of a WMN is that its topology, in terms of active nodes and links, can vary quite much. OLSR is adapted to this feature by continuously updating the information that any node has about any other node, based on the most recent ‘scanning’ of the network. It thus finds good-enough routes to all destinations.

Previously, our goal was to model OLSR and analyse its properties [15, 17, 18]. There are numerous frameworks and techniques, formal and less formal, that one can choose for modelling purposes. Since we are interested in analysis, formal methods with their underlying mathematical foundations are best suited. However, the question is which formal method to choose. In this paper we resume our experiences with two formal methods, the Uppaal model checker and the Event-B theorem prover.

In Uppaal [7], safety and liveness properties are expressed using Computation Tree Logic (CTL). Constants, data structures and procedures are defined in a C-like language and modularity is addressed via components, represented as timed automata, that communicate with each other via channels. Uppaal has a model checking tool¹, that supports the basic computational model and checks whether properties hold for a model or not, in the latter case providing a counterexample. In Event-B [2], safety properties are expressed in first-order logic, while constants, data structures, variables and their updates are modelled in a guarded command language. Event-B has a theorem prover tool, the Eclipse-based Rodin platform², that supports the basic modelling and analysis, based on generating and discharging proof obligations. Modularity is addressed via refinement: a model is initially abstract and details are added to it in proof-safe manner. Liveness properties are modelled logically or with specific update types.

Contributions. After modelling and analysing OLSR with both Uppaal and Event-B, we found that both formal methods are useful, but at different scales and for emphasising different aspects of modelling and analysis. In this paper, our contribution is to provide a comparison of our respective models as well as of these formal methods, with suggestions for modellers as to when to use one or another. We take into account four main criteria w.r.t. our models (Uppaal and Event-B models) comparison: parts of the protocols that have been modelled, particular properties that have been verified, networks topologies that have been modelled and data structures that have been used when modelling. To overview the applicability of Uppaal and Event-B, we provide a comparison between them by focusing on their specification languages, their mechanism for variable updating, their modularity methods, their verification strategies, their scalability potentials and their real-time modelling capabilities. Based on our

¹ <http://uppaal.org/>.

² <http://www.event-b.org/>.

considerations, we provide several guidelines for when to use Uppaal or Event-B for formal modelling and analysis.

Outline. We proceed as follows.³ In Sect. 2 we describe in some detail the formal tools employed in the paper, namely Uppaal and Event-B. In Sect. 3 we overview the OLSR protocol and in Sect. 4 we summarise our modelling of OLSR in Uppaal and Event-B, respectively. In Sect. 5 we compare our Uppaal and Event-B models as well as the frameworks themselves. We draw some usage guidelines of these formal tools in practical situations in Sect. 6.

2 Formal Methods, Model Checking, and Theorem Proving

A formal method usually refers to a framework allowing one to model, analyse, verify, and animate a system. A formal method has a formal semantics based on mathematics, and can thus provide precise answers to questions about systems properties. A formal method includes a specification (or modelling) language, analysis methods, various modularity mechanisms addressing the scale of a system; nowadays, successful formal methods also have tools associated to them, including editors, analysers, animators, and more.

When modelling the dynamic behaviour of a system with a formal method, each execution step in the model follows from a semantical rule of inference and hence can be checked by a mechanical process. The advantage of formal methods is that they provide valuable means to symbolically examine the entire state space of a system model and establish a correctness or safety property that is true for all possible inputs. These methods have a great potential on improving the correctness and precision of design and development, as they produce reliable results. However, this is rarely done in practice today, except for safety critical systems. In the rather recent past, one of the reasons was the lack of user friendly and scaling tools, combined with the enormous complexity of real systems. Nowadays however, we have good tools for several formal methods, so one of the questions remaining for the adoption of formal methods in industry remains: which tool is more suitable for a certain (type of) system?

In this paper we set out to examine two different tools associated to two formal methods, namely model checking and theorem proving.

2.1 Model Checking—Uppaal’s Timed Automata

Model checking (e.g. [9]) is an algorithmic and automatic approach used to validate and verify key correctness properties in finite representations of a formal system model. By modelling the behaviour of a system in mathematical language, model checking exhaustively and automatically checks whether the model meets

³ The detailed descriptions of our models appear in [16] for the Uppaal model and in [19] for the Event-B model.

a given specification. In model checking, Temporal Logic (TL) is used to specify and check the correct behaviour of a system. One of the most used model checking tools nowadays is the Uppaal model checker.

Uppaal [7, 20] is an integrated model checker for modelling, simulating (validating) and verifying real-time systems. It is appropriate for systems that can be modelled as networks of timed automata extended with bounded integer variables, structured data types, functions and synchronisation channels. A timed-automata is a finite-state machine with clock variables that measure time progression. Each automaton can be represented as a graph consists of locations (optionally also consisting invariants) and edges between those locations having guards, synchronisation channels, and updates of some variables. A state of a system is defined by automata's locations, value of clocks, and the value of all local and global variables. An edge can be fired in an automaton which leads to a new state. This edge can be fired separately in the automaton or between different automata used for synchronisation.

Uppaal's verifier uses Computation Tree Logic (CTL) (e.g. [11]) to express system requirements (properties) offering two types of formulas: state formulas and path formulas. State formulas describe individual states of the model, whereas path formulas quantify over paths in the model.

2.2 Theorem Proving–Event-B

Event-B [2] is a formal technique based on the B-Method [1] and on the Action Systems [5] framework, provides means to model and analyse parallel, reactive and distributed systems. Rodin Platform [3] provides automated support for modelling and verifying such systems. Event-B uses two modules for defining system specifications and for expressing system properties, namely **context** and **machine**. A context consists of carrier sets and constants, and their properties are defined as axioms of the model. So, a context deals with the static part of the system whereas a machine contains the dynamic part of the system. A machine can access the contents of a context which is defined by the keyword **Sees** determining the relationship between the machine and the context.

A machine expresses the model state using **variables** that are updated by **events**. Events can have **guards** that need to evaluate to **true**, allowing the event to be executed. When having several events enabled simultaneously, one event is selected non-deterministically. A machine may also contain **invariants**, i.e., properties which must hold for any reachable state in the model. In other words, invariants must be satisfied before and after the occurrence of all events.

The **refinement** is the main developing strategy in Event-B where a machine, let's say machine A, is refined by another machine, let's say machine B, i.e., $A \sqsubseteq B$. This happens when A's behaviour is not altered by B in any way and more new variables are added in B as well as new events to update the new variables. This type of refinement employed for our modelling is called **superposition** refinement. In order to prove that machine B is the refinement of machine A, a set of so-called **proof obligations** is generated by the Rodin platform. Some of

these proof obligations are discharged automatically by Rodin and some require interactive discharging with the help of the modeller.

3 An Overview of Optimised Link State Routing

The Optimised Link State Routing (OLSR) is a proactive routing protocol developed for Mobile Ad-hoc Networks (MANETs) and Wireless Mesh Networks (WMNs). OLSR operates as a routing table-driven protocol; each node keeps information about all the other nodes of the network in order to transfer data packets from a source node to a destination node. Examples of information stored in the routing table of a node a are: to get to node b (from a) the next node to take is node c ; or, to get to node b from a takes n hops, where a, b, c are nodes in the network and n is a natural number. Keeping the information in the routing table up-to-date is realised by nodes periodically exchanging specific *control* messages. OLSR is an optimisation over other link state protocols, since it decreases the network traffic by restricting the broadcasting of control messages to only specific nodes.

OLSR works in a completely distributed manner and does not require any central entity for coordination. Each node selects a set of one-hop neighbour nodes that have links to the two-hop neighbours of that selector node. The selected nodes are called **MultiPoint Relays (MPRs)** and are allowed to transmit control messages intended for diffusion into the entire network. There are two types of control messages, namely **HELLO** and **TC (Topology Control)** messages.

HELLO messages are broadcast every 2s and are used to determine one-hop and two-hop neighbours of each node as well as to select **MPR** nodes. These messages are only broadcast on single hops (to one-hop neighbours) and are not forwarded. **TC** messages are broadcast every 5s for building and refreshing topological information in the routing tables. These messages are broadcast on single hops and can be forwarded through the network via **MPR** nodes. Upon receipt of **HELLO** or **TC** messages, the receiving node updates its routing table based on the information in the received control message. Therefore, the topological information is always kept up-to-date in the routing tables in order to deliver data packets to arbitrary destination nodes.

4 Formal Modelling of the OLSR

We now present the overview of our OLSR models, i.e., Uppaal and Event-B models of the OLSR protocol. Both formal models are described in detail in our technical reports [16, 19].

4.1 Uppaal Model of the OLSR

In [15], we modelled OLSR in Uppaal as a parallel composition of identical processes, each indicating the behaviour of each node of the network. Every

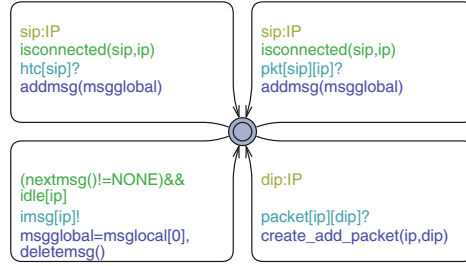


Fig. 1. Overview of model development

process is itself a parallel composition of two timed-automata, i.e., **OLSR** and **Queue**. The **OLSR** automaton is modelling the complete behaviour of the routing protocol [10] and **Queue** automaton (depicted in Fig. 1) is chosen to model the input buffer of every node in the network.

Nodes are able to broadcast and handle different types of messages (**HELLO**, **TC** and **PACKET**) in the network (modelled by **OLSR**) and the connected neighbour nodes can receive the incoming messages and store these messages in their input buffer (modelled by **Queue**). Whenever the **OLSR** is ready to handle a message (is not busy) and there are messages stored in the **Queue**, the **OLSR** and the **Queue** synchronise together on the `img` channel, moving a message from the **Queue** to the **OLSR** for processing.

The **OLSR** models the routing table of a node using a local data structure. Routing tables provide all the necessary information to route data packets to different destination nodes. Connectivity between two nodes is modelled by the predicate `isconnected[i][j]`, denoting a node-to-node communication. If two nodes are in transmission range of each other, they can communicate with each other via channels. In order to model rigorous timing behaviour, we defined several clocks for each **OLSR** to model on-time broadcasting control messages, to consider time spent to send every message, and to update and refresh the information in the routing tables.

Based on [10], each node in the network broadcasts a **HELLO** message every 2s containing the information about the originator of the message and the one-hop neighbours of the **HELLO** message originator. Upon receipt of a **HELLO**, the receiving node updates its routing table for the **HELLO** message originator and its two-hop neighbours (one-hop neighbours of the **HELLO** message originator). The receiving node also selects its **MPR** nodes which are able to broadcast **TC** messages through the network. Such nodes (**MPRs**) then broadcast **TC** messages every 5s through the network. **TC** messages contain the information about the originator of the **TC** messages, **MPR** nodes of the message originator, etc. When a node receives a **TC** message, it first checks if the message is considered for processing following some conditions. If so, then the receiving node updates its routing table for the **TC** message originator and the **MPR** nodes of the **TC** originator. Afterwards, if the receiving node is an **MPR** and the **TC** message is considered for forwarding, the **TC** is forwarded to the next nodes.

The **Queue** (Fig. 1) models storing incoming messages from other nodes (directly connected neighbour nodes) of the network. The incoming messages are buffered and in turn are sent to the **OLSR** for further processing. Messages can be received only if the receiving node is connected to the sender of the message. In this case, the **Queue** of the receiving nodes stores the messages to its local data queue.

4.2 Event-B Model of the OLSR

In [18], we developed a formal model of the OLSR protocol at five different levels of abstraction (depicted in Fig. 2) using Event-B (Rodin platform). We have defined two contexts containing constants and carrier sets, whose properties are expressed as a list of axioms for the model. These contexts contain the static part of the system. The dynamic part of our system is modelled using five machines that describe the state of the model with their variables which are updated by events. These five machines are related to the contexts and can access them using the keyword *sees* as shown in Fig. 2. Also, the more abstract machines and contexts are refined into more concrete machines and contexts using keywords ‘refines’ and ‘extends’, respectively.

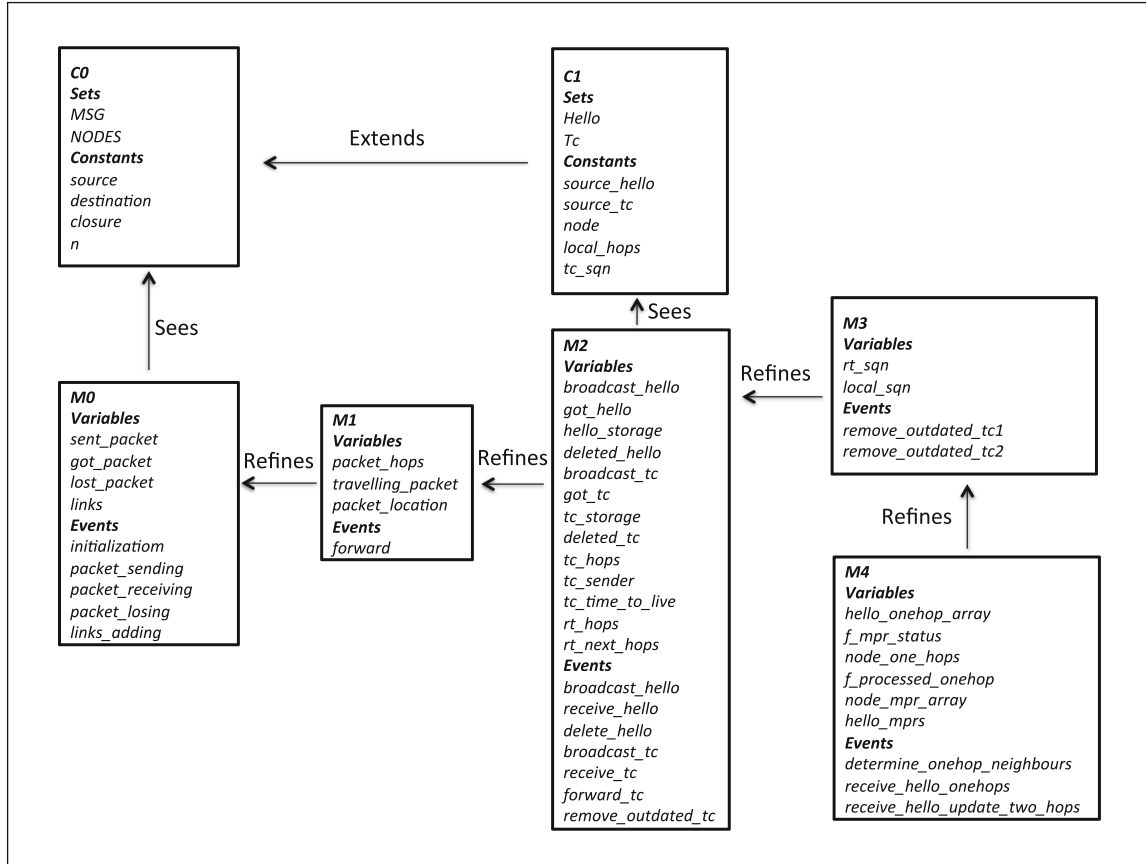


Fig. 2. Overview of model development

Our initial model M0 deals with basic protocol behaviour, i.e., sending, receiving, and losing data packets as well as an abstraction of proactive routing behaviour (adding links between nodes). First refinement M1 models a storing and forwarding architecture when data packets are transferred hop by hop from a source node to a destination node. Second refinement M2 models the basic behaviour of the route discovery protocol, describing the OLSR behaviour when sending and receiving control messages as well as updating routing tables. Third refinement M3 models how the protocol decides to process only new control messages and how to avoid processing control messages with old information. Fourth refinement M4 models the selection of MPR nodes, helping to decrease the traffic in the network.

```

Event   packet_receiving  $\hat{=}$ 
any
where   msg
then   grd1 :  $msg \in sent\_packet \setminus (got\_packet \cup lost\_packet)$ 
end     act1 :  $got\_packet := got\_packet \cup \{msg\}$ 

```

In M0, data packets are received from a source node to a destination node in an atomic step which is of course not the case in reality. In real protocols, data packets are forwarded hop by hop from a source node to a destination node using multi-hop communication that is modelled in the more concrete machine M1. For instance, event *packet_receiving* models the successful receiving of the data packet *msg* by a destination node. The guard of this event (*grd1*) models that *msg* has not been received or lost yet. When the packet is received, it will be added in the *got_packet* set.

In M1, the storing and forwarding architecture of data packets is modelled while all nodes are not connected and the data packets must be forwarded hop by hop through the destination. In this step, we model a local storage for each node to store these incoming packets and forward these data packets to next nodes along the path to the destination node.

In M2, nodes are able to broadcast and handle different types of messages (HELLO, TC and PACKET) in the network (modelled by several events). Also routing tables of nodes are modelled as variables, providing the information to deliver data packets to different destination nodes. Every node broadcasts a HELLO message having the information only about the HELLO message originator. Upon receipt of a HELLO message, the corresponding routing table for the originator of the HELLO message is updated. Also, each node broadcasts a TC message containing the information about the TC message originator, number of hops of the TC message, sender of the TC message and time to live of the TC message (number of hops that a TC message can be forwarded). Upon receipt of a TC message, the corresponding routing table for the originator of the TC message is updated and if the TC message is considered for forwarding, it is forwarded to the next nodes.

In M3, we extend the routing table of every node and also add a new variable in the TC message in order to model sequence numbers. Sequence numbers are embedded in TC messages to avoid processing messages with old information.

Also, we defined several events to update the local sequence number of each node and to remove out-dated messages from the network.

In M4, we restrict the broadcasting of TC messages to only specific nodes, namely MPRs, and not all nodes broadcast TC messages through the network. We added one-hop neighbours of the HELLO message originator in the HELLO messages so that upon receipt of a HELLO message, the two-hop neighbours of the receiving nodes can be also updated. In this case, nodes can determine their MPR nodes and also nodes are able to recognise whether or not they are MPR nodes of some other nodes in the network. If some nodes are selected to be MPRs, then they can broadcast/forward TC messages through the network.

5 Comparison

In this section, we compare our OLSR models, the Uppaal model [15] and the Event-B model [18] as well as the modelling tools Uppaal [7] and Event-B [2].

5.1 Uppaal Model vs Event-B Model

Table 1 depicts an overview of our comparison. We take into the account four main criteria: what parts of the protocol we've modelled, what properties we've verified for our models, for what types of network topologies we modelled the protocol and what data structures we've used.

Table 1. Overview of our models comparison

	Uppaal model	Event-B model
Protocol	Core functionality	Core functionality with timing abstraction
Properties	Route establishment packet delivery non-optimal route finding recovery time	Route establishment packet delivery non-optimal route finding
Topologies	All topologies up to 5 nodes	All topologies with n nodes
Data Structures	Queues	Relations, functions

Protocol. We were able to model the core functionality of the OLSR protocol [10] in both Uppaal and Event-B. This functionality refers to the behaviour that is always required for the protocol to perform. The only feature that we abstracted away in our Event-B model was the timing of messages. In the OLSR protocol [10], HELLO and TC messages are sent periodically. We have abstracted away the treatment of time in Event-B as this is still incipient, involving a rather different perspective of treating variables as continuous functions of time [4, 6].

Table 2. Overview of Uppaal and Event-B comparison

	Uppaal	Event-B
Specification Language	Timed automata, C-like language	Set theory, guarded commands language
Variables Update	Transition: selection guard update	Event: parameter guard action
Modularity	Divided into several automata at the same level of abstraction	Divided into several machines at different levels of abstraction
Verification	CTL automatically providing counterexamples	First-order logic automatically and interactively no counterexamples
Scalability	Small-scale systems (finite)	Large-scale systems (infinite)
Real Time	Precisely models timing variables	Partially models timing variables

Properties. We verified our OLSR model in Uppaal for the following properties: route establishment, packet delivery, optimal route finding, and recovery time. We were able to verify that all nodes in the network can establish routes to different destination nodes as well as deliver data packets to these destinations. We proved by finding a counterexample that OLSR is not always able to find optimal routes to all the destinations as well as showed that OLSR needs a relatively long time to recover after a link breakage in the network [15]. In our Event-B model, we verified our OLSR model for the following properties: route establishment, packet delivery and optimal route finding. We came to the same conclusions as for our Uppaal model. Routes are established to all destinations and data packets are delivered to these destinations; however, these routes may be non-optimal w.r.t. the hop counts. Since we abstracted away from timing properties, we did not investigate the recovery time of OLSR in Event-B.

Topologies. We verified our Uppaal model of OLSR for all network topologies up to 5 nodes. Since the model checking technique suffers from the state space explosion problem, we were not able to extend our analysis for more realistic networks. However, when modelling in Event-B, we were not restricted by the number of nodes in the network and we could verify the protocol for arbitrary networks with n number of nodes.

Data Structures. We modelled the OLSR protocol in Uppaal and Event-B with different data structures. In our Uppaal model, we have defined the **Queue** timed automata to store different types of incoming messages to a node. In Event-B, we modelled the storing architecture using relations between nodes and messages. We defined a specific data structure in Uppaal to model the routing tables,

whereas in Event-B we defined different variables to model routing tables. The types of nodes in the network were defined by integers in Uppaal, while in Event-B we introduced a carrier set to model the network nodes. We defined a common data structure for all types of messages in Uppaal; in Event-B, we introduced different carrier sets for each type of message. We note here that we can have the same data structure (modelling all types of messages, i.e., data packets and control messages) also in Event-B, and this is part of some future generalisation that we plan for modelling various network protocols in Event-B.

5.2 Uppaal vs Event-B

Table 2 depicts an overview of the comparison between Uppaal and Event-B. We detail this table below, namely we compare the specification languages, the variable updating mechanisms, the modularity methods, the verification strategies, the scalability potential, and the real-time modelling capabilities.

Specification Language. The Uppaal model checker uses *timed-automata* as the specification language whereas Event-B is based on *set theory*. In Uppaal, constants, data structures and procedures are defined in a C-like language. In Event-B, constants, data structures, variables and their updates are modelled in a guarded command language.

Variable Updating Mechanism. In Uppaal *transitions* are used to update the variables while in Event-B *events* accomplish the same thing. In both formal methods, the state of the model is determined by the values of the variables. We show the similarities between transitions in Uppaal and events in Event-B by sketching an example based on our models when a node receives a message as depicted in Fig. 3. By this, we also demonstrate how our models in Uppaal and Event-B are equivalent. These similarities are as following:

- *Selection of Parameters.* In Uppaal, the **select** label of a transition consists of a list of **name:type** expressions, where **name** is the variable's name and **type** is its type. As depicted in Fig. 3 (Transition), IP is the type of variable **ip**, i.e., an integer in our model. This variable is only accessible for the respective

<p>Transition in Uppaal:</p> <pre> - select ip:IP - guard msglocal.msgtype == packet - update delivered++ </pre>	<p>Event in Event-B:</p> <pre> - any msg - where msg ∈ sent_packet \ (got_packet ∪ lost_packet) - then got_packet := got_packet ∪ {msg} </pre>
--	--

Fig. 3. Transition and event in Uppaal and Event-B.

- transition and it takes a non-deterministic value in the range of its respective types (integer type in our model). In Event-B ([Event](#)), the **any** clause of an event lists the parameters (or local variables) of the event, i.e., `msg` in Fig. 3; the types of these parameters are usually specified in the guards of the events.
- *Guards.* In Uppaal, the **guard** label refers to logical expressions that determine if the respective transitions are enabled (when guards hold). In Fig. 3 ([Transition](#)), `msglocal.msgtype == packet` is the guard of the transition and shows if the received message is a new packet. In Event-B, the **where** clause contains the guards of the events, i.e., the logical conditions for the event to be enabled (when guards hold). The guard of the ([Event](#)) in Fig. 3 is shown as `msg ∈ sent_packet \ (got_packet ∪ lost_packet)`.
 - *Updates and Actions.* In Uppaal, the **update** label of a transition contains a list of expressions that update the values of variables. In Fig. 3 ([Transition](#)), `delivered++` is the update that increases the value of integer variable `delivered` showing that the packet has been received. In Event-B, the **then** clause lists the actions of the event that modify some variables of the model. In Fig. 3 ([Event](#)), `got_packet := got_packet ∪ {msg}` is the action that adds the receiving packet to the received messages set. In both frameworks, the variable updating mechanism takes place only if the guards of transitions or events respectively hold.

Modularity. In order to model the whole system’s behaviour in Uppaal, several automata are introduced, each modelling different parts of the system. These automata need to synchronise with each other, to keep the consistency and relevance between different parts of the system model. However, it is not always possible to split the system into different automata and thus a system model may remain too complex to understand, having too numerous transitions. In Event-B, different machines are introduced to fully model the behaviour of the system at different levels of abstraction, starting from a very simple and abstract level. This abstract model is stepwise developed using refinement methods to finally model the complete behaviour of the entire system. Consistency between the different levels of refinements is verified by discharging proof obligations. The stepwise development allows to split the complexity of the system into different levels and makes it easier to understand the model and discharge the proof obligations.

Verification. In Uppaal, the required properties are expressed in Computational Tree Logic (CTL) syntax and the whole system model is verified for the defined properties. In Event-B, invariants are used to formulate system properties using first-order logic; the invariants have to be checked for the whole system in order to show the consistency between different levels of abstractions. Properties in Uppaal are discharged fully automatically whereas in Event-B some of the properties are discharged automatically and some are discharged interactively. Uppaal provides counterexamples if a property does not hold; this helps in finding errors in the system. In Event-B, if a proof obligation is not discharged automatically, this typically signals some modelling problem and the modeller is prompted back to remodel certain aspects.

Scalability. Uppaal, like all model checking tools, suffers from the state space explosion problem, hence it is not able to verify very large and complex systems. Event-B allows to verify even large and complex systems. Event-B checks the general validity of a property for *all* models (i.e., also for infinite models) whereas Uppaal is dedicated to small-scale, finite systems.

Real-Time. Uppaal provides clock variables to model timing behaviour of real-time systems whereas for Event-B modelling timing behaviour is still incipient. In Uppaal, clock variables model discrete timing behaviour. In Event-B, advances are made to model hybrid behaviour including discrete and continuous time modelling [4,6], but these are not implemented in the Rodin platform yet. In Event-B, the time can be defined as a function that can be mapped to an integer variable increasing by the events.

6 Conclusions and Usage Guidelines

To resume our experiences of modelling OLSR with Uppaal and Event-B, we essentially found that the two formalisms require different approaches to modelling. In Uppaal, the modeller attempts to capture the whole system, in all its complexity, from the beginning, aided in this task by the modularity technique of splitting the model into communicating time automata. In Event-B, the modeller gets to understand the system's complexity by modelling it in increasingly more detailed levels of abstraction. When we have a conceptually complex system (behaviour of routing protocols), choosing Uppaal or Event-B for modelling it and analysing it is ultimately a matter depending on the modeller's experience.

One can specify properties to prove in both formalisms, but the verification of these properties differs in the two frameworks. In Uppaal, the verification depends on the size of the model and may be unsuccessful if the size is bigger (networks of realistic size) than some arbitrary and typically small value. This is because model checking enforces a brute force verification of properties in all possible states of the system, thus leading relatively fast to overflow. Approaches are taken to overcome this problem, such as partial order reduction techniques [21] and statistical model checking. The former assumes that not all states are worth verifying, and thus defines a priority-based order relation that imposes the verification of the most important states only. The latter employs probabilities and gives results such as the property holds with a 0.99 probability; these probabilities are calculated based on many random walks through the state space (simulations of) the system. In Event-B, the verification of properties is based on logic and proof engines that are built to work for any defined mathematical concepts, including infinite-sized models. When properties are not verified automatically, Uppaal provides counterexamples exposing the offending state: this can be quite useful for correcting errors. In the same situation, the Rodin platform shows the unsatisfied proof obligation and thus the modeller gets some feedback on what does not work. We note here that, if there are flaws in the system, often they are exposed even for small-scale models, see [12,15].

Both Uppaal and Event-B are supported by performant software platforms for modelling and proving; depending on how advanced these platforms are, some aspects can be modelled or not, such as real-time properties. Uppaal was designed to include clock variables and time modelling, while Event-B was designed as a general refinement-based framework. We can precisely model real-time properties of communication protocols in Uppaal, e.g., broadcasting a control message at a certain time. Recently, several approaches were proposed on how to add real-time modelling in Event-B in a conservative manner, e.g. Hybrid Event-B [6] or [4]. This would imply that all variables except clocks are functions of time, so a slight change of perspective is needed here. Real-time properties are typically closely related to implementation details, for instance, to various network parameters; hence, even if we can model timing, when translating the final model into a software product, we might need to alter various properties and parameters anyway.

For modelling and verifying routing protocols, Uppaal remains very useful, as it provides synchronisation mechanisms used in wireless networks: broadcast and binary synchronisation. This allows to closely understand the communication between network nodes. Besides these clear differences, we found that modelling in either framework is quite natural and rewarding and, once the modeller is experienced enough with the framework, quite efficient as well.

To the best of our knowledge, this is the first paper comparing Uppaal and Event-B with respect to what each can model and prove. Relations between model checking and theorem proving in general have been studied before, e.g. [13], where for solving a (rather simple) puzzle, arguments are given for using model checking instead of theorem proving. We note that real systems are very complex nowadays and thus, proving properties for the system, independently of its size, is quite important. Another interesting observation made in [13] is that theorem proving helps in constructing the model, while model checking can be used when we already understand the model quite well. Other approaches connecting model checking and theorem proving are [8], where the idea is to combine the two methods and more recently [24], where refinement is studied in the context of both Uppaal and Event-B.

References

1. Abrial, J.R.: *The B-book: Assigning Programs to Meanings*. Cambridge University Press, New York (1996)
2. Abrial, J.R.: *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, New York (2010)
3. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. *STTT* **12**(6), 447–466 (2010)
4. Abrial, J.-R., Su, W., Zhu, H.: Formalizing hybrid systems with Event-B. In: Derrick, J., Fitzgerald, J., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) *ABZ 2012*. LNCS, vol. 7316, pp. 178–193. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-30885-7_13](https://doi.org/10.1007/978-3-642-30885-7_13)

5. Back, R.J.R., Sere, K.: From action systems to modular systems. In: Naftalin, M., Denvir, T., Bertran, M. (eds.) FME 1994. LNCS, vol. 873, pp. 1–25. Springer, Heidelberg (1994). doi:[10.1007/3-540-58555-9_83](https://doi.org/10.1007/3-540-58555-9_83)
6. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core hybrid Event-B I: single hybrid Event-B machines. *Sci. Comput. Program.* **105**, 92–123 (2015)
7. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). doi:[10.1007/978-3-540-30080-9_7](https://doi.org/10.1007/978-3-540-30080-9_7)
8. Berezin, S.: Model checking and theorem proving: a unified framework. Ph.D. thesis, Carnegie Mellon University (2002)
9. Clarke, E.M., Emerson, E.A., Sifakis, J.: Model checking: algorithmic verification and debugging. *Commun. ACM* **52**(11), 74–84 (2009)
10. Clausen, T., Jacquet, P.: Optimized link state routing protocol (OLSR). RFC 3626 (Experimental) (2003). <http://www.ietf.org/rfc/rfc3626>
11. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, vol. B. Formal Models and Semantics, pp. 995–1072. MIT (1995)
12. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Modelling and analysis of AODV in UPPAAL. In: 1st International Workshop on Rigorous Protocol Engineering, pp. 1–6 (2011)
13. Halpern, J., Vardi, M.: Model checking vs. theorem proving: a manifesto. In: Lifschitz, V. (ed.) Artificial Intelligence and Mathematical Theory of Computation, pp. 151–176. Academic Press Professional, Inc. (1991)
14. Johnson, D., Hu, Y., Maltz, D.: The Dynamic Source Routing Protocol (DSR). RFC 4728 (Experimental) (2007). <http://www.ietf.org/rfc/rfc4728>
15. Kamali, M., Höfner, P., Kamali, M., Petre, L.: Formal analysis of proactive, distributed routing. In: Calinescu, R., Rumpe, B. (eds.) SEFM 2015. LNCS, vol. 9276, pp. 175–189. Springer, Cham (2015). doi:[10.1007/978-3-319-22969-0_13](https://doi.org/10.1007/978-3-319-22969-0_13)
16. Kamali, M., Kamali, M., Petre, L.: Formally analyzing proactive, distributed routing. Technical report 1125, TUCS - Turku Centre for Computer Science (2014)
17. Kamali, M., Petre, L.: Improved recovery for proactive, distributed routing. In: 20th International Conference on Engineering of Complex Computer Systems (ICECCS 2015), pp. 178–181. IEEE (2015)
18. Kamali, M., Petre, L.: Modelling link state routing in Event-B. In: 21st International Conference on Engineering of Complex Computer Systems, ICECCS 2016, pp. 207–210. IEEE (2016)
19. Kamali, M., Petre, L.: Modelling link state routing in Event-B. Technical report 1154, TUCS - Turku Centre for Computer Science (2016)
20. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. *Int. J. Softw. Tools Technol. Transf. (STTT)* **1**(1), 134–152 (1997)
21. Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Compact data structures and state-space reduction for model-checking real-time systems. *Real-Time Syst.* **25**(2–3), 255–275 (2003)
22. Neumann, A., Aichele, C., Lindner, M.: Better Approach To Mobile Ad-hoc Networking Routing Protocol (B.A.T.M.A.N.). IETF Draft (2008). <https://tools.ietf.org/id/draft-openmesh-b-a-t-m-a-n-00.txt>
23. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc On-Demand Distance Vector Routing Protocol (AODV). RFC 3561 (Experimental) (2003). <http://www.ietf.org/rfc/rfc3561>
24. Vain, J., Tsiopoulos, L., Bostrom, P.: Integrating refinement-based methods for developing timed systems. In: Petre, L., Sekerinski, E. (eds.) From Action Systems to Distributed Systems: The Refinement Approach, pp. 171–185. CRC Press (2016)

Paper V

AODVv2: Performance vs. Loop Freedom

Mojgan Kamali, Massimo Merro and Alice Dal Corso

Originally published in: *proceedings of the 44th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2018)*, Lecture Notes in Computer Science, pp. 337-350, Springer, 2018.

©2018 Springer. Reprinted with permission from the publisher.

AODVv2: Performance vs. Loop Freedom

Mojgan Kamali¹() , Massimo Merro²() , and Alice Dal Corso²

¹ Faculty of Science and Engineering, Åbo Akademi University, Turku, Finland

`mojgan.kamali@abo.fi`

² Dipartimento di Informatica, Università degli Studi di Verona, Verona, Italy

`massimo.merro@univr.it`

Abstract. We compare two evolutions of the Ad-hoc On-demand Distance Vector (AODV) routing protocol, i.e. DYMO and AODVv2-16. In particular, we apply *statistical model checking* to investigate the performance of these two protocols in terms of routes established and looping routes. Our modelling and analysis are carried out by the Uppaal Statistical Model Checker on 3×3 grids, with possibly lossy communication.

1 Introduction

Ad hoc networking has gained popularity and is applied in a wide range of applications, such as public safety and emergency response networks. Mobile Ad-hoc Networks (MANETs) are self-configuring networks that support broadband communication without relying on wired infrastructure. *Routing protocols* of ad-hoc networks are among the main factors determining performance and reliability of these networks. They specify the way of communication among different nodes by finding appropriate paths on which data packets must be sent.

In this work, we focus on two evolutions of the Ad-hoc On-demand Distance Vector (AODV) [21] protocol to investigate their performance and to analyse if they may yield routing loops. The protocol finds alternative routes *on demand* whenever needed, meaning that it is intended to first establish a route between a source node and a destination (*route discovery*), and then maintain a route between the two nodes during topology changes (*route maintenance*).

Most studies of wireless network protocols, especially for large scale networks, are mostly done by *simulation techniques* and test-bed experiments. These are valuable techniques for performance analysis, however they do not allow us to simulate all possible scenarios. As a consequence, unexpected behaviours and flaws appear many years after the development of protocols. Formal analysis techniques allow to screen protocols for flaws and to exhibit counterexamples to diagnose them. For instance, *model checking* [6] provides both an exhaustive search of all possible behaviours of the system, and exact quantitative results.

Statistical Model checking (SMC) [25] is a technique combining model checking and simulation, aiming at providing support for quantitative analysis as well as addressing the size barrier to allow analysis of large models. It relies on choosing sampling traces of the system and verifying if they satisfy the given property

with a certain probability. In contrast to exhaustive approach, statistical model checking does not assure a 100% correct result, but it is possible to restrict the probability of an error occurring. In this work, we apply Uppaal SMC [8], the statistical extension of the Uppaal model checker [2] to support the composition of timed and/or probabilistic automata. In Uppaal SMC, two main statistical parameters α and ϵ , in the interval $[0, 1]$, must be specified by the user; the number of necessary runs is then computed by the tool using the Chernoff-Hoeffding bounds. The tool provides a value in the confidence interval $[p-\epsilon, p+\epsilon]$ indicating the probability p of the intended property. Parameters α and ϵ represent the probability of *false negatives* and probabilistic *uncertainty*, respectively.

Since its first definition, AODV has seen several versions and improvements. In particular, DYMO [22] is an evolution of AODV supporting *path accumulation*: whenever a control message travels via more than one node, information about all intermediate nodes is accumulated in the message and distributed to its recipients [7]. Several studies have shown that AODV, DYMO and AODVv2 suffer from *routing loops* [5, 10, 14, 20], i.e. an established route stored in the routing tables at a specific point in time that visits the same node more than once before the intended destination is reached [11]. Caught packets in a routing loop can saturate the links and decrease the network performance. Thus, loop freedom is a critical and challenging property for any routing protocol.

Contributions. Our work has been motivated by a recent version of AODVv2, appearing in the AODVv2-16 Internet draft [24] and containing a number of modifications to overcome the looping problem of AODV and DYMO. As a first contribution, we have modelled in Uppaal SMC the core functionality of both AODVv2-16 and DYMO protocols for 3×3 grid topologies (9 nodes). While the model for AODVv2-16 is completely new, the model for DYMO is a refinement of those appearing in [7, 15]. In both cases, we have adopted a *probabilistic model* for wireless communication to take into account both *message loss* and *link breakage* at different rates. As a second contribution, we have compared the performance of DYMO and AODVv2-16 with respect to four different workbenches: (i) route discovery, (ii) number of routes found, (iii) optimal route finding, (iv) and packet delivery. From our analysis, it emerges that DYMO performs significantly better than AODVv2-16 with respect to all workbenches, in particular in the presence of a significant message loss rate. Finally, as the third contribution, we investigate whether the models for the two protocols may yield routing loops under extreme conditions, such as message loss and link breakage. As expected, our model of DYMO faces a number of loops; however the corresponding Uppaal model for AODVv2-16 is loop free, with an accuracy of 99%, suggesting that the changes introduced in this version of the protocol help to reduce/remove loops.

Outline. In Sect. 2, we overview both DYMO and AODVv2-16. In Sect. 3, we discuss the Uppaal models of the two protocols based on their RFCs [22, 24]. In Sects. 4 and 5, we present the results of our analysis with respect to performance and loop occurrences. In Sect. 6, we draw conclusions and review related work.

2 DYMO and AODVv2-16: Two Evolutions of AODV

This section provides a brief overview of DYMO and AODVv2-16 protocols. In both protocols, each node maintains a *routing table* (*RT*) containing information about the routes to be followed when sending messages to the other nodes of the network. The collective information in the nodes' routing table is at best a partial representation of network connectivity as it was at some times in the past; in the most general scenario, mobility together with node and communication failures continually modify that representation.

We report a scheme of the DYMO protocol [22] with an injected packet having the source node s and destination node d . When s receives the data packet, it first looks up an entry for d in its routing table. If there is no such entry, it broadcasts a **rreq** message through the network. Afterwards when an intermediate node receives the **rreq**, it first checks whether or not the information in the message is new. If this is not the case, the receiving intermediate node discards the **rreq** and the processing stops. If the information is new, the receiving node updates its routing table based on the information in the **rreq**. Then, it checks if it has a route to the destination d . If this information is provided, intermediate node sends a **rrep** back to the source s as well as to the destination d . By this, DYMO establishes *bidirectional* routes between originator and destination. On the other hand, if the intermediate node does not have any route to d , it adds its own address to the **rreq** and rebroadcasts the message.

When next intermediate node receives the rebroadcast **rreq**, it updates (if the message is new) the routing table entry associated with s and the corresponding intermediate sender node and repeats the same steps executed by the former intermediate node. Finally when the destination d receives the **rreq**, it updates its routing table for the source node s and all the intermediate nodes that have rebroadcast the **rreq**, and then sends a unicast **rrep** that follows the reverse path towards s . Each node receiving the **rrep** will update the routing table entry associated with d and intermediate nodes.

Nodes also monitor the status of alternative *active* routes to different destinations. Upon detecting the breakage of a link in an active route, an **rerr** message is broadcast to notify the other nodes about the link failure. The **rerr** message contains the information about those destinations that are no longer reachable toward the broken link. When a node receives an **rerr** from its neighbours, it invalidates the corresponding route entry for the unreachable destinations.

The architecture of the AODVv2-16 protocol [24] is quite similar to that of DYMO considering some differences. One of the main differences of AODVv2-16 is to avoid sending **rrep** by intermediate nodes. When AODVv2-16 broadcasts a **rreq**, it waits to get the **rrep** back only from the destination of the **rreq**. It means that intermediate nodes do not send the **rreps** to the source of the **rreq** even if they have active routes through the destination node. This behaviour will increase the time needed for route discovery (routing tables in AODVv2-16 are not updated as often as in DYMO), decreasing the performance of the protocol¹.

¹ Due to lack of space, we highlight the design differences between two protocols in [16].

2.1 Degrading Performance to Avoid Routing Loops

Different studies have proved the presence of loops in AODV, DYMO and AODVv2 protocols [5, 10, 14, 20]. Here, we report a simple example to show how a loop can occur in DYMO, and how this is avoided in AODVv2-16.

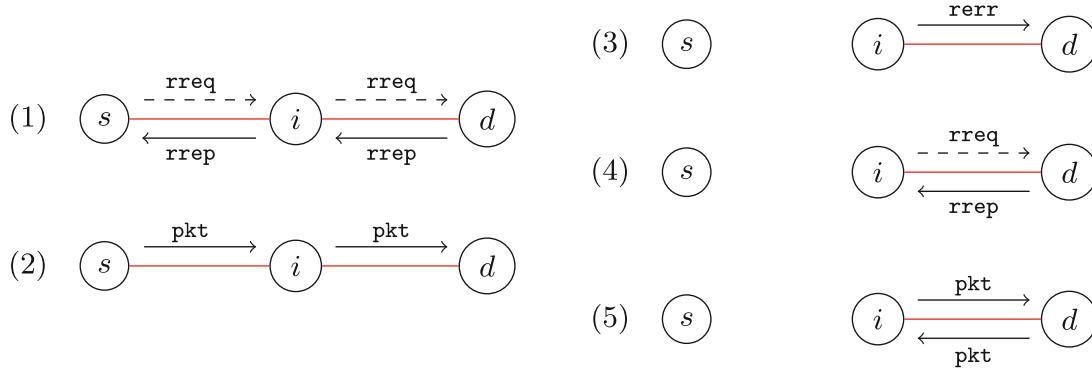


Fig. 1. Presence of a loop in DYMO.

The network in Fig. 1 consists of three nodes that are connected in a linear topology. Let's assume that node *s* has a *pkt* to send to node *d*. It initiates the route discovery and broadcasts a *rreq*. Node *i* gets the *rreq*, updates its routing table for node *s*, adds itself as an intermediate node in the *rreq* of *s*, and rebroadcasts the *rreq*, Fig. 1(1). Node *s* and *d* receive the *rreq*. Node *s* drops the message since the received message is its own *rreq* and node *d* updates its routing table for node *s* and *i* and since it is the *rreq* destination, it sends a *rrep* back through the path to the originator of the *rreq*, i.e. node *s*. Node *i* gets the *rrep* from *d*, updates its routing table for *d*, adds itself as an intermediate node in *rrep* of *d* and sends the *rrep* to *s*. Finally, node *s* receives the *rrep* of *d*, Fig. 1(1), updates its routing table for *i* and *d* and sends the *pkt* to node *i* to be delivered to *d*, Fig. 1(2).

Afterwards, the link between *s* and *i* breaks and node *i* has a *pkt* to send to *s*. Node *i* becomes aware of the link breakage and broadcasts an *rerr* to its neighbours. Assume the *rerr* from *i* is lost in the reception of *d*, resulting in node *d* not being notified about the link breakage, Fig. 1(3). Next when node *i* has another *pkt* to send to *s*, and it knows already that there is no valid route to *s*, it initiates a *rreq* to its neighbours. Node *d* receives the *rreq* and it has the valid route to *s*. Node *d*, as the intermediate node, sends the *rrep* to *i*, Fig. 1(4). Node *i* receives the *rrep* from *d* and updates its routing table for node *s* with new information. In this situation, node *i* sends its *pkt* to *d* since node *i*'s next hop through *s* is *d*. Node *d* then sends the *pkt* to *i* as node *d*'s next hop through *s* is *i*. Finally, the *pkt* is circulated in a loop, Fig. 1(5).

Protocol designers have overcome the looping problem of DYMO by incorporating several changes in the new version (AODVv2-16). In this current version, if route discovery is initiated the intermediate nodes which have active routes

through the destination do not send the **rrep** to the originator, meaning that the destination of the **rreq** has sole responsibility for sending the **rrep** back to the originator. By this, they have solved the problem of having loops in the network, but the performance level has decreased.

In AODVv2-16, the routing tables can be updated if:

- “If AdvRte is more recent than all matching LocalRoutes. ”
- “If the sequence numbers are equal, Check that AdvRte is safe against routing loops compared to all matching LocalRoutes, If LoopFree(AdvRte, LocalRoute) returns TRUE, compare route costs:
 - If AdvRte is better than all matching LocalRoutes, it MUST be used to update the Local Route Set because it offers improvement.
 - If AdvRte is not better (i.e. it is worse or equal) but LocalRoute is Invalid, AdvRte SHOULD be used to update the Local Route Set because it can safely repair the existing Invalid LocalRoute.” [[24], page 28]

Here, **LocalRoutes** stores previously received messages, **AdvRte** contains the information about newly received message, and $\text{LoopFree}(\text{AdvRte}, \text{LocalRoute}) := (\text{Cost}(\text{AdvRte}) \leq \text{Cost}(\text{LocalRoute}))$.

There are more conditions in the specification of the AODVv2-16 indicating when to update routing tables, leading to less information being stored, hereby decreasing the performance. For instance, routing tables in AODVv2-16 are not updated in the scenario where sequence numbers are the same, the message is received via a longer path, and the link toward a destination is broken, although updating would have helped to fix broken paths. In addition, the sending of **rrep** by intermediate nodes is not specified in AODVv2-16. This leads to routes being established more slowly than in DYMO, since the **rreq** has to travel all the way to the destination node and **rrep** has to be sent back along the whole path, from the **rreq** destination to the **rreq** originator.

3 Uppaal Models of AODVv2-16 and DYMO

In this section, we briefly explain our AODVv2-16 automata and provide some modifications of the Uppaal SMC model of [15] for DYMO². As in [15], both protocols are represented as parallel compositions of node processes, where each process is a parallel composition of two timed automata, the **Handler** and the **Queue**. This is because each node maintains a message queue to store incoming messages and a process for handling these messages; the workflow of the handler depends on the type of the message. Communication between nodes *i* and *j* is only feasible if they are neighbours, i.e. in the transmission range of each other. This is modelled by predicates of the form **isconnected**[*i*][*j*] which are true if and only if *i* and *j* can communicate. Communication between different nodes *i* and *j* are on channels with different names, according to the type of the control message being delivered (**rrep**, **rreq**, **rerr**).

Messages (arriving from other nodes) are stored in the queue, by using a function **addmsg()**. Only messages sent by nodes within the transmission range

² The reader can consult our models at <http://users.abo.fi/mokamali/SOFSEM2018>.

may be received. Unlike the model of [15] our **Queue** is essentially a probabilistic timed automata. Uppaal SMC features branching edges with associated weights for the probabilistic extension. Thus we define an integer constant **loss**, with $0 \leq \text{loss} \leq 100$, and a node can either lose a message with weight **loss** or receive it with weight $(100 - \text{loss})$.

The **Handler** automaton, modelling the message-handling protocol, is far more complicated and has around 22 locations. The implementation of the two protocols differs for this automaton. The **Handler** is busy while sending messages, and can only accept one message from the **Queue** once it has completely finished handling the previous message. Whenever it is not processing a message and there are messages stored in the **Queue**, the **Queue** and the **Handler** synchronise via channel **imsg[ip]**, transferring the relevant message data from the **Queue** to the **Handler**. According to the specification of the protocols, the most time consuming activity is the communication between nodes, which takes 40 ms on average [22, 24]. This is modelled in the **Handler** by means of a clock variable **t**, set to 0 before transmission, so that a delay between 35 and 45 ms is selected uniformly at random.

Based on DYMO and AODVv2-16 specifications, **rreqs** can be resent the maximum of 3 times in the presence of message loss. The major differences between AODVv2-16 and DYMO, are the absence of intermediate **rreps** and also conditions regarding updates of the routing tables. As we explained in Sect. 2, AODVv2-16 tries to find the whole path through the destination node and it does not rely on the **rreps** from intermediate nodes that have routes through the destination node (intermediate nodes do not generate any **rrep** message even if they have active routes through the destination node).

Finally, we report the main changes which have been introduced in our Uppaal SMC model of DYMO with respect to that proposed in [15]:

- In the DYMO model by [15], two connected nodes could get disconnected while a node is waiting to transmit a message (waiting time of 40 ms), which could cause a potential deadlock in the system. For our experiments, we modify this behaviour and assume that two connected nodes cannot get disconnected during this period of time which is the case in reality (the probability that two nodes disconnect upon communication is too low).
- We minimised the DYMO automaton of [15] by removing a number of redundant locations and transitions that were modelling the same procedure.
- We have also modelled the resending of **rreq** for the maximum number of 3 times, when control messages, i.e. **rreq**, **rrep** and **rerr**, can get lost. This was done by adding new locations and transitions.
- In the current version of DYMO Uppaal model, when a node receives a message from its neighbour it first checks the message sequence number. If it is recent then it updates its routing table for the message originator and for the stored intermediate nodes in the message. If the sequence number is not recent, the message is simply dropped without any routing table update.

For further details the reader is referred to our technical report [16].

4 Performance Analysis on Static Grids

We replay the experiments of [7, 15] to compare DYMO and AODVv2-16 on 3×3 grid topologies with possibly lossy channels. Furthermore, we investigate one more property, namely *packet delivery*. More precisely, we consider four different workbenches to compare the two protocols: 1. A probabilistic analysis to estimate the ability to successfully complete the protocol finding the requested routes for a number of properly chosen scenarios; 2. A quantitative analysis to determine the average number of routes found during the routing process in the same scenarios; 3. A qualitative analysis to verify how good (i.e. short) are the routes found by the routing protocol. 4. A probabilistic analysis to investigate the number of delivered packets to their corresponding destinations. We conduct our experiments using the following set-up: (i) 2.3 GHz Intel Quad-Core i7, with 16 GB memory, running the Mac OS X 10.11.6 “El Capitan”; (ii) Uppaal SMC model-checker 64-bit version 4.1.19. The statistical parameters of false negatives (α) and probabilistic uncertainty (ϵ) are both set to 0.01, leading to a confidence level of 99%. For each experiment with these parameters, Uppaal SMC checks several hundred runs of the model, up to 26492 runs (cf. Chernoff-Hoeffding bound). We run our experiments for the message loss rates used in [7], namely 0%, 10% and 30%, and then also for 40% to obtain more precise results.

4.1 Successful Route Requests

In the first set of experiments we consider four specific nodes: A, B, C and D; each with particular originator/destination roles. Our scenarios are a generalisation of those of [15] (as we consider larger networks) and assign roles as follows:

- (i) A is the only originator sending a packet first to B and afterwards to C;
- (ii) A is sending to B first and then B is also sending to C;
- (iii) A is sending to B first and then C is sending to D.

Up to symmetry, varying the nodes A, B, C and D on a 3×3 grid, we have 5184 different configurations. From this number we deduct 4518 configurations because they make little sense in our analysis, as the source and the destination node coincide. This calculation yields 666 different configurations. As we will repeat our simulations for four different loss rates, this makes in total 2664 experiments.

Initially, for each scenario no routes are known, i.e. the routing tables of each node are empty. Then, with a time gap of 35–45 ms, two of the distinct nodes receive a data packet and have to find routes to the packet’s destinations. The query in Uppaal SMC syntax has the following shape:

```
Pr[<=10000](<>(tester.final && emptybuffers() &&
art[0IP1][DIP1].nhop!=0 && art[0IP2][DIP2].nhop!=0))
```

The first two conditions require the protocol to complete; here, `tester` refers to a process which injects to the originators nodes (`tester.final` means

that all data packets have been injected), and the function `emptybuffers()` checks whether the nodes' message queue are empty and the `Handler` is idle (is not busy with processing messages). The third and the fourth conditions require that two different route requests are established. Here, `art[o][d].nhop` is the next hop in `o`'s routing table entry for destination `d`. As soon as this value is set (is different to 0), a route to `d` has been established. Thus, the whole query asks for the probability estimate (Pr) satisfying the CTL-path expression `<>(tester.final && emptybuffers() && art[OIP1][DIP1].nhop!=0 && art[OIP2][DIP2].nhop!=0)` within 10000 time units (ms); as in [15] this bound is chosen as a conservative upper bound to ensure that the analyser explores paths to a depth where the protocol is guaranteed to have terminated.

In Table 1 we provide the results of our query for both models. More precisely, we report the average probability to satisfy the required property in all 666 configurations. This is done for four different loss rates. Note that in the case of perfect communication, our analysis shows that the probability to successfully establish a required route in our setting can be estimated to be at least 0.99. We should add here that increasing message loss rate leads an increase in the number of runs to complete the simulation. This is because unreliable communication channels make the routing process longer in order to resend control messages. In other words, the number of runs is affected by the lower success probability which requires a larger number of runs to provide confidence intervals.

Table 1. Route establishment on 3×3 grid networks ($\alpha = \epsilon = 0.01$).

	Loss = 0%	St. dev.	Loss = 10%	St. dev.	Loss = 30%	St. dev.	Loss = 40%	St. dev.
DYMO	0.99	0.00	0.99	0.00	0.89	0.06	0.65	0.14
AODVv2-16	0.99	0.00	0.98	0.00	0.72	0.14	0.45	0.20

We can see that on the 3×3 grid with perfect communication the reliability of the two protocols is quite similar. However, in the presence of message loss, DYMO performs better than AODVv2-16. In fact, the higher the loss rate, the bigger the gap between the two protocols. More precisely, with a 10% loss rate DYMO performs better than AODVv2-16, whereas with 30% and 40% loss rate the gap between two protocols becomes more obvious (DYMO performs much better than AODVv2-16). It should be also noticed that the results of the simulations on DYMO are more homogeneously distributed around the average probability, as it appears from the smaller standard deviation.

4.2 Number of Route Entries

The second analysis proposed in [15] takes into account the capability to build other routes while establishing a route between two specific nodes. Routing tables are updated whenever control messages are received. Both protocols update for

the whole discovered paths by forcing *path accumulation* (storing the information about intermediate nodes in control messages).

We check the property:

$$E[\leq 10000, 26492] (\text{max:total_knowledge}())$$

where the function `total_knowledge()` counts the number of non-empty entries appearing in all routing tables built along a run of the protocol, and the function `max` returns for all runs of the simulation, the maximum number of non-empty entries. This calculation is done for all different configurations; the result of the analysis is the average over all configurations. The reader should notice that this kind of query is different from the previous one. It has the form $E...$, where the letter “E” stands for *expected value estimation*, as the result of the query is a value and not a probability. Since the number of runs is not determined by value estimation, we set 26492 runs for our simulations to guarantee a 99% confidence level. The time bound remains as 10000.

Table 2. Route quantity on 3×3 grid networks (26492 runs for each experiment).

	Loss 0%	St. dev.	Loss 10%	St. dev.	Loss 30%	St. dev.	Loss = 40%	St. dev.
DYMO	37.27	7.68	37.42	6.18	34.68	5.86	31.27	5.39
AODVv2-16	34.01	5.93	34.38	5.76	34.57	5.91	31.66	5.36

We repeat the same analysis of [15] on our 3×3 grid by considering four different loss rates. In total we did 2664 experiments, one for each configuration with a different loss rate. The results of our analysis are reported in Table 2. Table 2 shows that during the routing process DYMO establishes more routes than AODVv2-16 (37 versus 34 routes), in the absence of message loss. This gap remains the same when having 10% message loss rate. The analysis shows that increasing the rate of the message loss leads to have similar behaviour of DYMO and AODVv2-16 (having the same number of route entries).

4.3 Optimal Routes

The results of the previous section tell us that in our 3×3 grid, DYMO is more efficient than AODVv2-16 in populating routing tables while establishing routing requests. In this section, we provide a class of experiments to compare the ability of two protocols in establishing optimal routes, i.e. routes of minimal length, according to the network topology. As explained in [15, 19], all ad-hoc routing protocols based on `rreq`-broadcast can establish non-optimal routes. This phenomenon is more evident in a scenario with unreliable communication.

We replay the same experiments of [15]. We checked the following property:

$$\text{Pr}[\leq 10000] (< (\text{tester.final} \ \&\& \ \text{emptybuffers}()) \ \&\& \ \text{art}[OIP1][DIP1].\text{hops} == \text{min_path} \ \&\& \ \text{art}[OIP2][DIP2].\text{hops} == \text{min_path1})).$$

Here, the third and the fourth conditions require that two different route requests are established. In fact, `art[o][d].hops` returns the number of hops necessary to reach the destination node `d` from the originator `o`, according to `o`'s routing table. Furthermore, we require this number to be equal to the length of the corresponding optimal route (which has been previously computed).

In this experiment we are not interested in checking all non-empty routing entries but only those which are directly involved in the two routing requests. This property is checked on all 666 configurations with four different loss rates. Notice that this time we ask for a probability estimation, so the result is going to be a probability. The statistical parameters of our simulations are $\alpha = \epsilon = 0.01$.

Table 3. Optimal routing on 3×3 grid network. ($\alpha = \epsilon = 0.01$).

	Loss 0%	Stand. dev.	Loss 10%	Stand. dev.	Loss 30%	Stand. dev.	Loss = 40%	Stand. dev.
DYMO	0.94	0.20	0.84	0.18	0.67	0.17	0.48	0.17
AODVv2-16	0.95	0.19	0.86	0.18	0.58	0.19	0.37	0.19

Table 3 says that the probability to establish optimal routes in the two routing protocols is very close when having no message loss. Actually, in the presence of message loss, there is still a gap in favour of DYMO. This gap would become bigger if we would focus only on the optimality of the second route request, which is launched slightly after the first one. This is because DYMO works better than AODVv2-16 when routing tables are not completely empty.

4.4 Packet Delivery

The packet delivery property differs from the successful route request property, in that the route establishment property only checks if the source node has the information about the destination node, however the packet delivery property checks if the injected packets are delivered to the destination at the end. Indeed, there might be a situation where an originator node has the information about the destination node and sends its packet to the next node along the path to the destination node, but the next node itself does not have valid information about the destination node. As a consequence, all the packets stemming from the originator node will be lost, hence the packets cannot arrive at the destinations.

This property in Uppaal SMC syntax is as following:

```
Pr[<=10000](<>(tester.final && emptybuffers() && empty_queues()==0 &&
packet_delivered()==2))
```

Here, the third and the fourth conditions require that the two packets are delivered at their destinations; `empty_queues()` is a function checking whether or not there is any packet in the queue of any nodes. When this function returns 0, it shows that there is no more packet in the queues of nodes. Function `packet_delivered()` returns the number of delivered packets which must be 2 at the end, given that we have injected two packets for our experiments. Thus, the whole query asks for the probability estimate (Pr) satisfying the CTL-path

expression `<>(tester.final && emptybuffers() && empty_queues()==0 && packet_delivered()==2)` within 10000 time units (ms); as in [15] this bound is chosen as a conservative upper bound to ensure that the analyser explores to a depth where the protocol is ensured to have terminated.

The results in Table 4 show that AODVv2-16 works worse than DYMO w.r.t. the packet delivery property as it tries to find the whole path to the destination node, whereas DYMO relies on replying back from the intermediate nodes. Moreover, routing tables in AODVv2-16 are not updated regularly due to the more restricted routing table updates in AODVv2-16. Therefore, the probability that all packets are delivered to the destination nodes is lower in AODVv2-16.

5 Loop Analysis on Grids with Link Breakage

We run our experiments, looking for loops on 3×3 grids during the routing process, under the assumption that links between nodes can break with a *high probability*. We model link breakage by modifying the **Queue** automaton so that when a control message is received by the queue of a node (using a function `addmsg()`) with probability of `100-loss`, the link between one random node in the network and the receiver can break with a fixed probability `breaks`. Since link breakage is one of the main factors causing routing loops, we assign this value to 80, so that with a very high probability the link between the sender and the receiver fails. Furthermore, in order to increase the traffic in the network we inject *three packets* in total. The slightly new scenario is explained below.

Table 4. Packet delivery on 3×3 grid networks ($\alpha = \epsilon = 0.01$).

	Loss 0%	Stand. dev.	Loss 10%	Stand. dev.	Loss 30%	Stand. dev.	Loss=40%	Stand. dev.
DYMO	0.99	0.00	0.98	0.00	0.78	0.09	0.50	0.16
AODVv2-16	0.99	0.00	0.97	0.01	0.60	0.16	0.35	0.18

We consider again four specific nodes: A, B, C and D; each with particular originator/destination roles. We assign roles as follows: (i) A is the only originator sending the first packet to B, and afterwards sends the second and third packets to C; (ii) A is sending to B first and then B is also sending the second and third packets to C; (iii) A is sending to B first and then C is sending the second and third packets to D.

For simplicity, in order to work with a reasonable number of experiments, second and third packets have the same originators and destinations, so the number of configurations up to symmetry will remain the same, i.e. 666. In our experiments we check the number of loops in all 666 different configurations (how many loops exist in the network) and we show how many configurations have routing loops i.e. in how many configurations an injected packet can be circulated between nodes. This gives 2664 experiments in total for each protocol. Our experiments can be represented using the following Uppaal SMC syntax:

`E[<=10000;26492] (max:numberofloops())`

Function `numberofloops()` counts the number of loops found along a run of the protocol, and the function `max` returns for all runs of the simulation, the maximum number of loops. We maintain the same number of runs as for performance analysis, i.e. 26492, to guarantee a 99% accuracy.

Table 5 depicts the maximum number of loops considering different message loss rate in different configurations for both protocols. The results of our analysis show that when message loss rate increases, the number of loops in the networks for DYMO also increases. For instance when having 0% message loss, the number of loops in the network is 1 and when message loss increases to 10% or more number of loops in the network increases to 2. Unlike DYMO, the rate of message loss does not have any effect on the number of loops in the network for AODVv2-16 as we cannot find routing loops while verifying AODVv2-16.

Table 5. Number of loops in different configurations.

	Loss 0%	Loss 10%	Loss 30%	Loss 40%
DYMO	1	2	2	2
AODVv2-16	0	0	0	0

Table 6. Number of configurations that have loops.

	Loss 0%	Loss 10%	Loss 30%	Loss 40%
DYMO	10	11	13	11
AODVv2-16	0	0	0	0

Table 6 shows the number of configurations having loops. Results for DYMO show with 0% message loss there are 10 configurations out of 666 that have loops in the network. This value is increased to 11 with 10% message loss, and when message loss is increased to 30%, the number of configurations that have loops goes up to 13. The table depicts when message loss increases to 40%, the number of configurations that have loops decreases to 11. In contrast to DYMO, there is no configuration in AODVv2-16 that has routing loops.

6 Conclusions and Related Work

Our work has been strongly inspired by recent version of AODVv2-16 [24] where several modifications were proposed to overcome looping problem of DYMO (and previous versions of AODVv2). We believe that the protocol designers accepted the performance hit in order to ensure that the protocol is loop free. To the best of our knowledge, our work is the first to investigate the looping property of AODVv2-16 and compare the performance of DYMO and AODVv2-16.

In this paper, we modelled the AODVv2-16 protocol and investigated the performance of the protocols DYMO and AODVv2-16 in 3×3 grids, with possibly lossy communication, as well as checking the loop freedom property for both protocols. Our analysis is performed using the Uppaal SMC (release 4.1.19). We were able to show how the performance of the more recent AODVv2-16 has been worsened compared to DYMO, especially in the case of lossy communication. DYMO can cause routing loops whereas our extensive analysis was not able to find loops in AODVv2-16. This result encourages us to pursue towards a formal proof of loop freedom for AODVv2-16.

Formal analysis of MANETs and their protocols is a challenging task, and their formal verification have attracted the attention from formal methods community [1, 3, 4, 7, 13, 15, 17, 18, 20]. There are number of papers which apply (statistical) model checking to AODV and its variants, to test the performances of the protocol(s). Fehnker et al. [9] used the Uppaal model checker [2] to analyse basic qualitative properties of the AODV routing protocol in all network topologies up to five nodes. Höfner and McIver [15] showed that AODV performs better than DYMO on the same topologies, relying on the Uppaal SMC model checker. On the contrary, Dal Corso et al. [7] showed that on larger networks (4×3 toroids) with lossy communication DYMO performs better than AODV.

There are also several studies on loop freedom of AODV and DYMO. van Glabbeek et al. [14] have studied the loop freedom of the AODV protocol and they have showed that AODV is not loop free and sequence numbers do not guarantee loop freedom. Namjoshi and Treffler [20] have investigated the looping property of AODVv2-04 and they have proved this protocol causes routing loops. There are several other studies that confirm existence of routing loops in AODV [5, 10, 12]. In a recent paper, Yousefi et al. [26] have applied their extension of actor-based modelling language bRebeca to model AODVv2-11 [23] (a previous version of AODVv2) where they have proved that the loop freedom property of AODVv2-11 does not hold. The authors had reported the existing loop scenario to protocol designers and the protocol has been modified in the newer version (AODVv2-13).

References

1. Battisti, L., Macedonio, D., Merro, M.: Statistical model checking of a clock synchronization protocol for sensor networks. In: Arbab, F., Sirjani, M. (eds.) FSEN 2013. LNCS, vol. 8161, pp. 168–182. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40213-5_11
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
3. Benetti, D., Merro, M., Viganò, L.: Model checking ad hoc network routing protocols: ARAN vs. endairA. In: SEFM 2010, pp. 191–202. IEEE (2010)
4. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. *J. ACM* **49**(4), 538–576 (2002)
5. Bres, E., van Glabbeek, R., Höfner, P.: A timed process algebra for wireless networks with an application in routing. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 95–122. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49498-1_5
6. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
7. Dal Corso, A., Macedonio, D., Merro, M.: Statistical model checking of Ad Hoc routing protocols in lossy grid networks. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 112–126. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17524-9_9

8. David, A., Larsen, K.G., Legay, A., Mikušionis, M., Poulsen, D.B.: Uppaal SMC tutorial. *STTT* **17**(4), 397–415 (2015)
9. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: Flanagan, C., König, B. (eds.) *TACAS 2012*. LNCS, vol. 7214, pp. 173–187. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_13
10. Fehnker, A., van Glabbeek, R.J., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. *CoRR* abs/1312.7645 (2013)
11. Garcia-Luna-Aceves, J.J.: A unified approach to loop-free routing using distance vectors or link states. *SIGCOMM Comput. Commun. Rev.* **19**(4), 212–223 (1989)
12. Garcia-Luna-Aceves, J.J., Rangarajan, H.: A new framework for loop-free on-demand routing using destination sequence numbers. In: *MASS 2004*, pp. 426–435. IEEE (2004)
13. van Glabbeek, R., Höfner, P., Portmann, M., Tan, W.L.: Modelling and verifying the AODV routing protocol. *Distrib. Comput.* **29**(4), 279–315 (2016)
14. van Glabbeek, R., Höfner, P., Tan, W.L., Portmann, M.: Sequence numbers do not guarantee loop freedom: AODV can yield routing loops. In: *MSWiM 2013*, pp. 91–100. ACM (2013)
15. Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: Brat, G., Rungta, N., Venet, A. (eds.) *NFM 2013*. LNCS, vol. 7871, pp. 322–336. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_22
16. Kamali, M., Merro, M., Dal Corso, A.: AODVv2: performance vs. loop freedom. Technical report. pp. 1177. TUCS - Turku Centre for Computer Science (2016)
17. Kamali, M., Höfner, P., Kamali, M., Petre, L.: Formal analysis of proactive, distributed routing. In: Calinescu, R., Rumpe, B. (eds.) *SEFM 2015*. LNCS, vol. 9276, pp. 175–189. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22969-0_13
18. Merro, M., Ballardin, F., Sibilio, E.: A timed calculus for wireless systems. *TCS* **412**(47), 6585–6611 (2011)
19. Miskovic, S., Knightly, E.W.: Routing primitives for wireless mesh networks: design, analysis and experiments. In: *INFOCOM 2010*, pp. 1–9. IEEE Press (2010)
20. Namjoshi, K.S., Treffer, R.J.: Loop freedom in AODVv2. In: Graf, S., Viswanathan, M. (eds.) *FORTE 2015*. LNCS, vol. 9039, pp. 98–112. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19195-9_7
21. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (AODV) Routing. RFC 3561 (Experimental) (2003)
22. Perkins, C., Stan, R., Dowdell, J.: Dynamic MANET on-demand (AODVv2) Routing draft-ietf-manet-dymo. Internet Draft 26 (2013)
23. Perkins, C., Stan, R., Dowdell, J., Steenbrink, L., Mercieca, V.: Ad Hoc On-demand Distance Vector (AODVv2) Routing draft-ietf-manet-aodvv2. Internet Draft 11 (2015)
24. Perkins, C., Stan, R., Dowdell, J., Steenbrink, L., Mercieca, V.: Dynamic MANET On-demand (AODVv2) Routing draft-ietf-manet-aodvv2. Internet Draft 16 (2016)
25. Sen, K., Viswanathan, M., Agha, G.A.: Vesta: a statistical model-checker and analyzer for probabilistic systems. In: *QEST 2005*, pp. 251–252. IEEE (2005)
26. Yousefi, B., Ghassemi, F., Khosravi, R.: Modeling and efficient verification of wireless ad hoc networks. *CoRR* abs/1604.07179 (2016)

Paper VI

Adaptive Formal Framework for WMN Routing Protocols



Mojgan Kamali and Ansgar Fehnker

Originally published in: *proceedings of the 15th International Conference on Formal Aspects of Component Software (FACS 2018)*, Lecture Notes in Computer Science, pp. 1-21, Springer, 2018.

©2018 Springer. Reprinted with permission from the publisher.



Adaptive Formal Framework for WMN Routing Protocols

Mojgan Kamali¹() and Ansgar Fehnker²()

¹ Åbo Akademi University, Turku, Finland

² University of Twente, Enschede, The Netherlands
mojgan.kamali@abo.fi, ansgar.fehnker@utwente.nl

Abstract. Wireless Mesh Networks (WMNs) are self-organising and self-healing wireless networks that provide support for broadband communication without requiring fixed infrastructure. A determining factor for the performance and reliability of such networks is the routing protocols applied in these networks. Formal modelling and verification of routing protocols are challenging tasks, often skipped by protocol designers. Despite some commonality between different models of routing protocols that have been published, these models are often tailored to a specific protocol which precludes easily comparing models. This paper presents an adaptive, generic and reusable framework as well as crucial generic properties w.r.t. system requirements, to model and verify WMN routing protocols. In this way, protocol designers can adapt the generic models based on protocol specifications and verify routing protocols prior to implementation. This model uses Uppaal SMC to identify the main common components of routing protocols, capturing timing aspect of protocols, communication between nodes, probabilities of message loss and link breakage, etc.

1 Introduction

Wireless Mesh Networks (WMNs) are self-organising and self-healing wireless networks that provide support for broadband communication without requiring fixed infrastructure. They provide rapid and low-cost network deployment and have been applied in a wide range of application areas such as public safety, emergency response networks, battlefield areas, etc.

A determining factor for the performance and reliability of such networks is the routing protocols applied in these networks. Routing protocols specify the way of communication among nodes of the network and find appropriate paths on which data packets are sent. They are grouped into two main categories: proactive and reactive routing protocols. Proactive protocols rely on the periodic broadcasting of control messages through the network (time-dependent) and have the information available for routing data packets. Reactive protocols, in contrast, behave on-demand, meaning that when a packet targeting some destination is injected into the network they start the route discovery process.

Previous studies of routing protocols mostly rely on simulation approaches and testbed experiments. These are appropriate techniques for performance analysis but are limited in the sense that it is not possible to simulate systems for all possible scenarios. Formal techniques, mathematically languages and approaches, are used to complement testbed experiments and simulation approaches. They provide tools to design and verify WMN routing protocols, allow to model protocols precisely and to provide counterexamples to diagnose their flaws.

Statistical Model checking (SMC) combines model checking with simulation techniques to overcome the barrier of analysing large systems as well as providing both qualitative and quantitative analysis. Uppaal SMC monitors simulation traces of the system and uses sequential hypothesis testing or Monte Carlo simulation (for qualitative and quantitative analysis respectively) to decide if the intended system property is satisfied with a given degree of confidence. Statistical model checking does not guarantee a 100% correct result, but it is able to provide limits on the probability that an error occurs. In this work, we apply Uppaal SMC [7], the statistical extension of Uppaal.

Contributions. Our work has been inspired by the fact that formal modelling and verification of routing protocols seem challenging tasks. Protocol designers often decide on skipping this level of development. We provide an adaptive, generic and reusable framework as well as crucial generic properties w.r.t. system requirements, to model and verify WMN routing protocols. In this way, protocol designers can adapt the generic models based on protocol specifications and verify routing protocols prior to implementation.

In particular, this study describes how to build reusable components within the constraints imposed by the Uppaal modelling language. It identifies the main components that routing protocols have in common, and how to map them to data structures, processes, channels, and timed automata in the Uppaal language. We show the validity and applicability of our models by modelling Better Approach To Mobile Ad-hoc Networking (BATMAN) [19], Optimised Link State Routing (OLSR) [5], and Ad-hoc On-demand Distance Vector version2 (AODVv2) [21] protocols using our framework.

Outline: The paper is structured as follows: in Sect. 2, we give an overview of the formal modelling language used in this paper. Then in Sect. 3, we shortly overview the general structure of WMN routing protocols. Section 4 is the core of this paper where we discuss our generic Uppaal framework as well as our generic Uppaal properties. Section 5 demonstrates the adaptability of our framework, sketching examples of BATMAN, OLSR, and AODVv2 protocols. We discuss related work in Sect. 6 and draw conclusions as well as propose future research directions in Sect. 7.

2 Modelling Language

Most routing protocols of WMNs have complex behaviour, e.g., real-time behaviour, and formal modelling and verification of such systems end up being

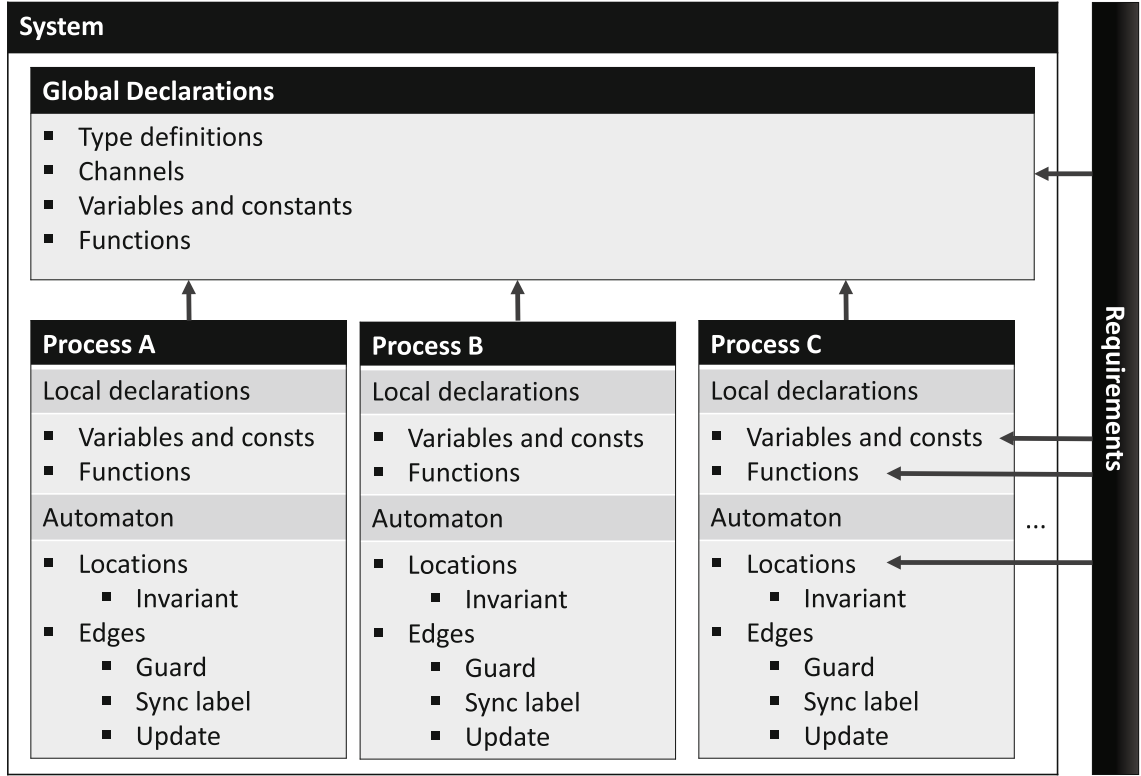


Fig. 1. Common structure of Uppaal models. Arrow denote access to variables and functions. Location names are treated like Booleans by the requirements.

a challenging task. Moreover, choosing a formal modelling language which considers the significant characteristic of these protocols is an important step in the development of a formal framework. In this work, we apply Uppaal SMC (which is based on stochastic time automata) to be able to realistically model timing behaviour, wireless communication, probabilistic behaviour, and complex data structure of routing protocols. In addition, the Uppaal GUI and Uppaal simulator provide a visualised interpretation of the system which makes the task of modelling easier. We describe the basic definitions that are used in Uppaal SMC.

2.1 Uppaal Timed Automata

The theory of timed automata [1] is applied for modelling, analysing and verifying the behaviour of real-time systems. A finite timed automaton is defined as a graph consisting of finite sets of locations and edges (transitions), together with a finite set of clocks having real values. The logical clocks of automata are initialised with zero and are increased with the same rate. Each location may have an invariant, and each edge may have guards (possibly clock guards) which allow a transition to be taken, and/or actions that can be updates of some variables and/or clocks.

The modelling language of Uppaal extends timed automata as defined by Alur and Dill [1] with various features, such as types, data structures, etc [2]. A system is a network of timed automata that can synchronise on channels

and shared variables. Fig. 1 depicts the common structure of Uppaal models. Uppaal distinguishes between global declarations and processes. All processes are running concurrently at the same level, and the model has no further hierarchy. Processes are actually instantiations of parameterised templates. Separate from the system model are the requirements, which describe properties, or statistical experiments.

Type definitions in the global declaration are used to define ranges of integers – often used as identifiers – or structs. Variables can be any integer type, any newly defined type, channels, and arrays of these. Clock variables that evaluate to real numbers are used to measure time. All clocks progress at the same rate, and can only be reset to zero. The global declaration can also define functions in a C-like language. These functions can be used anywhere in the model.

Each process has its own local declarations. These may contain variable declarations and function declarations. The scope of these is limited to the process. While it is possible to locally define types and channels, this is rarely done; at most to define urgent broadcast channels that force transitions once their guard becomes true.

For each process, there exists an automaton that operates on local and global variables and may use the locally and globally defined functions. Every automaton can be presented as a graph with locations and edges. Each location may have an invariant, and each edge may have a guard, a synchronisation label, and/or an update of some variables.

Synchronisation between automata happens via channels. For every channel a there is one label $a!$ to identify the sender, and $a?$ to identify receivers. Transitions without a label are internal; all other transitions use either binary handshake or broadcast synchronisation. Uppaal SMC supports only broadcast channels [7]:

Broadcast synchronisation means that one automaton with a $!$ -edge synchronises with several other automata that all have an edge with a relevant $?$ -label. The initiating automaton is able to change its location, and apply its update if and only if the guard on its edge is satisfied. It does not need a second automaton to synchronise with. Matching $?$ -edge automata must synchronise if their guards evaluate to true in the current state. They will change their location and update their states. First, the automaton with the $!$ -edge updates its state, then the other automata follow. If more than one automaton can initiate a transition on a $!$ -edge, the choice is made non-deterministically.

Due to the structure of the Uppaal model, automata cannot exchange data directly. A common workaround is the following: If an automaton wants to send data to another automaton, it synchronises on a channel. It writes the data to a global variable during an update, which is then copied by the second automaton to its local variable during its update.

Also, due to the scoping rules, one automaton cannot use a method of one of the other automata, for example, to query its state. The common workaround is to make either a duplicate of important information global, or to have the

information global, and have a self-imposed rule on which a process can read and write and to what part of the global variables.

In addition to the system model, it is possible to define requirements. Requirements can access all global and local variables, and use global and local functions, as long as they are side-effect free, i.e. they do not change variables outside of the scope of the function. Requirements can iterate over finite ranges, using `forall`, `exists`, or `sum` iterators.

Uppaal has several other keywords to define the behaviour of delays and transitions, such as `urgent` or `priority`. The discussion of these is outside of the scope of this paper. The common structure of Uppaal, with its scoping rules, however, is relevant for this paper, as it sets the framework in which we have to develop our generic model.

2.2 Uppaal Stochastic Timed Automata

Uppaal SMC [7] is a trade off between classical model checking and simulation, monitoring only some simulation traces of the system and uses sequential hypothesis testing or Monte Carlo simulation (for qualitative and quantitative analysis respectively) to determine whether or not the intended system property is satisfied with a given degree of confidence.

The modelling formalism of Uppaal SMC is based on the extension of Uppaal timed automata described earlier in this section. For each timed automata component, non-deterministic choices between several enabled transitions assigned by probability choices, refine the stochastic interpretation. A model in Uppaal SMC can consist of a network of stochastic timed automata that communicate via broadcast channels and shared variables.

Classical Uppaal's verifier uses a fragment of Computation Tree Logic (CTL) to model system properties. Uppaal SMC adds to its query language elements of the Metric Interval Temporal Logic (MITL) to support probability estimation, hypothesis testing, and probability comparison, and in addition the evaluation of expected values [7].

The algorithm for probability estimation [12] computes the required number of runs to define an approximation interval $[p - \epsilon, p + \epsilon]$ where p is the probability with a confidence $1 - \alpha$. The values of ϵ (probabilistic *uncertainty*) and α (*false negatives*) are selected by the user and the number of runs is calculated using the Chernoff–Hoeffding bound. In Uppaal SMC, the query has the form: $\text{Pr}[\text{bound}](\phi)$, where *bound* shows the time bound of the simulations and ϕ is the expression (path formula).

Evaluation of expected values of a max of an expression which can be evaluated to a clock or an integer value is also supported by Uppaal SMC. In this case, the *bound* and the number of runs (N) are given explicitly and then max of the given expression (*expr*) is evaluated. The query has the form: $\text{E}[\text{bound}; N](\max : \text{expr})$; an explicit confidence interval is also required for these type of queries.

Communication			
<i>global</i>	<ul style="list-style-type: none"> ▪ Message type definitions ▪ Channels for each message type ▪ Meta variables for exchanging values 		
Nodes		Topology	
Queue (proactive/reactive)		Handler (proactive/reactive)	
<i>global</i>	<ul style="list-style-type: none"> ▪ Channel for synchronizing with <i>Handler</i> 	<i>global</i>	<ul style="list-style-type: none"> ▪ Channel for synchronizing with <i>Queue</i> ▪ Routingtable
<i>local</i>	<ul style="list-style-type: none"> ▪ Buffer ▪ Methods for adding and deletion of messages 	<i>local</i>	<ul style="list-style-type: none"> ▪ Current message ▪ Clock for processing delay ▪ Methods update routing table ▪ Methods to create messages ▪ Methods to check whether message should be dropped.
<i>automaton</i>	<ul style="list-style-type: none"> ▪ Receiving messages ▪ Includes loss ▪ Passing to handler 	<i>local</i>	
Timers (optional)		Generator (proactive)	
<i>global</i>	<ul style="list-style-type: none"> ▪ Flag for restart and expired timers ▪ Urgent restart channel 	<i>automaton</i>	<ul style="list-style-type: none"> ▪ Receiving from <i>Queue</i> ▪ Dropping messages ▪ Processing, and sending of new messages.
<i>local</i>	<ul style="list-style-type: none"> ▪ Boolean for running timer ▪ Array of clocks 	<i>automaton</i>	<ul style="list-style-type: none"> ▪ Generate control messages at regular intervals.
<i>automaton</i>	<ul style="list-style-type: none"> ▪ Invariant for running timer ▪ Restart ▪ Expiring timer 	<i>automaton</i>	
Verification			
<i>global</i>	<ul style="list-style-type: none"> ▪ Variables for bookkeeping ▪ Methods for properties ▪ Methods to be used by tester 		
<i>local</i>	<ul style="list-style-type: none"> ▪ Variables for bookkeeping ▪ Methods for properties 		
<i>automaton</i>	<ul style="list-style-type: none"> ▪ Test automaton 		

Fig. 2. Generic structure of a WMN routing protocol model for verification, with respect to typical structure of an Uppaal model.

3 Overview Uppaal Model of WMN Routing Protocols

WMN routing protocols disseminate information in the network to provide the basis for selecting routes. Routing protocols specify which control messages should be sent through the network. These messages are received/lost by other network nodes. Receiving nodes update their information about other nodes based on received messages. Network nodes can send data packets to destinations in the network using discovered paths. Figure 2 depicts the main components that define a routing protocol model.

Communication. The protocol has to specify the types of control messages and the information they contain. This information consists of originator address, originator sequence number, etc. The model has to specify whether a message is sent as a unicast or as a multicast message¹.

Topology. The network topology shows how nodes are connected to each other. Connectivity is commonly modelled as an adjacency matrix. The model should provide methods to add and delete connections, as well as a model that determines how the topology changes. This paper uses a simple model of link failure; a more elaborate study of dynamic topologies can be found in [10].

¹ To avoid confusion, we will refer to this type of communication as *multicast*, instead of *broadcast*. We reserve the term *broadcast* for Uppaal channels.

Node. The behaviour of a protocol is defined by the composition of a queue, a handler, and – if the protocol is proactive – by one or more generators, and possibly a number of timers.

Queue. Messages from other nodes should be stored in a buffer or queue. Based on the order of arrival they will be processed later by the handler (first in, first out method is applied for buffers). If different messages arrive to a node at the same time, the choice of the message reception happens non-deterministically. It will use the information in these messages to update the corresponding routing information about sender nodes.

Handler. The handler is the core of the protocol. The handler will receive messages from the queue and update the stored information, which is kept in the routing table. Depending on the content of the message, and the current state of the routing table, the handler further decides whether to drop the message, send a broadcast message to all other neighbouring nodes, or send a unicast message to another node (both broadcast and unicast message consider sending delays).

Different ways of nodes communication (multicast and unicast) are modelled illustrated schematically in Fig. 3. For multicast synchronisation, we use an array of broadcast channels `Multicast[]`, one channel for each node. The invariant and the guard will encode the timing and duration of a transition (message delay). The guard of the corresponding edge of the receiving node – which will be part of its queue model – will encode the connectivity, i.e., it will not synchronise if the nodes are not connected. The sender will take the edge, regardless of whether other nodes are connected.

Unicast is modelled by a two-dimensional array of channels `Unicast[][]`, one channel for each pair of nodes. Unicast messages assume that on a lower level reception is acknowledged. If this fails, for example, if the nodes are not connected, the sender has to take an alternative transition. A typical alternative would be to multicast an error message or initiate a route request.

Generator. Proactive protocols send control messages at regular intervals. They highly depend on on-time broadcasting of their control messages in order to keep track of network information. Hence, each node includes also a model to generate those messages.

Timers. A protocol may use simple timers that can be reset, and expire after a set time. They can be used by the handler, to time delays or the duration of different modes of operation.

Verification. Since the model will be used for verification, the models will include parts that are only included for this purpose. This will include variables for bookkeeping and methods that check conditions on existing data structures. For this reason, the routing table of the handler was made global, to give access to verifications methods. Otherwise, the routing tables could be a local variable of the corresponding handler. The verification part of the model also often includes a test automaton, which may insert messages, change the topology, and record progress in response to certain events.

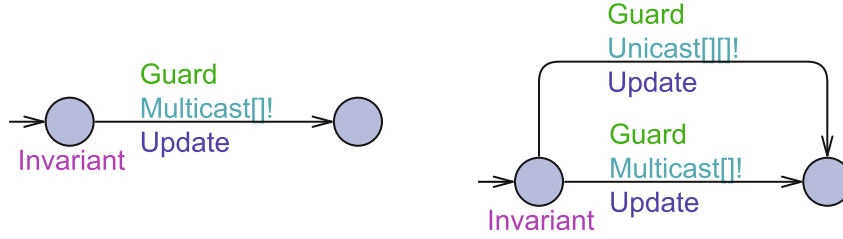


Fig. 3. Unicast and broadcast synchronisation.

The section omits the discussion of type definitions and constants that are used throughout the model. The next section will provide more detail on the various components of the Uppaal model.

4 Generic Uppaal Framework

Our framework consists of global declarations which are global in the system and accessible/updatable by all automata as well as local declarations that are exclusive to each automaton, i.e., these declarations can be accessed and updated only by the automaton itself. There are in total six templates for automata in our framework; four of them are used for modelling protocols and two are concerned with verification. The models are adaptable to protocol specifications, and they are available at <http://users.abo.fi/mokamali/FACS2018>.

4.1 Communication

To facilitate communication between network nodes in the model, there are a number of global declarations and type definitions. The number of nodes is `const int N`. Addresses of nodes are of type `typedef int[0, N - 1] IP`.

Communication can take place via unicast or multicast messages. The model includes the following channels:

```
broadcast chan unicast[N][N];
broadcast chan multicast[N];
urgent broadcast chan tau[N];
broadcast chan newpkt[N];
```

The `tau` channel is used to have internal transitions take place as soon as enabled. They are not used for synchronisation. The `newpkt` channel is used to insert a new packet at a given node.

A protocol must define for each type of message the message format. The reference implementation provides example for packets, route request messages, route reply messages, route error messages, and control messages, also known as TC messages. The format of a TC message, for example, is defined as:

```
typedef struct {
    IP oip;    //originator IP
```



```

int hops; //hops
TTLT ttl; //time-to-live
IP sip;    //sender IP
SQN osn;   //originator sequence number
} TCMSG;

```

The model will include a similar type definition for all types of messages. To make the treatment of message uniform we then define a generic message type as follows:

```

typedef struct {
    MSGTYPE msgtype; //Type of message
    TCMSG tc;         //TC message
    PACKET packet;    //Packet
    RREQMSG rreq;     //Route request msg
    RREPMSG rrep;     //Unicast route reply msg
    RERRMSG rerr;     //Route error msg
} MSG;

```

The field `msgtype` is an index into which type of message is being sent; only the corresponding field should be set. This construction is a work-around for not having *union types* in the Uppaal language.

Each type of message also comes with functions that generate a message of that type. They will be used for convenience and succinctness in the model. It also includes a global variable `MSG msgglobal`, which a sender copies into, and recipients copy from.

4.2 Topology

The network topology is defined by an adjacency matrix `topology[N][N]` with boolean type showing the directed connectivity between nodes, i.e., element 1 in the matrix shows that two nodes are directly connected and 0 indicates that two nodes are not connected directly, however they may be connected via some intermediate nodes. The connectivity between nodes is modelled by function `bool isconnected(IP i, IP j)`, and links can be dropped by calling function `void drop (IP i, IP j)`.

While protocols have to deal with mobility, the mobility models themselves are outside of the scope of this paper and these routing protocols. The processes that establish or delete links – and whether these processes are non-deterministic, stochastic, or probabilistic – are not part of the protocols themselves. The reference model includes a simple model `TopologyChanger` that drops between randomly selected nodes at a rate of 1 : 10. More elaborate models for changing topologies can be found in [10].

We should add here that even if we define a topology matrix to show the direct connectivity between nodes, the network is still wireless. It means that network nodes are not aware of each other before receiving the control messages, and they realise the connectivity only after they receive/process control messages from their neighbours.

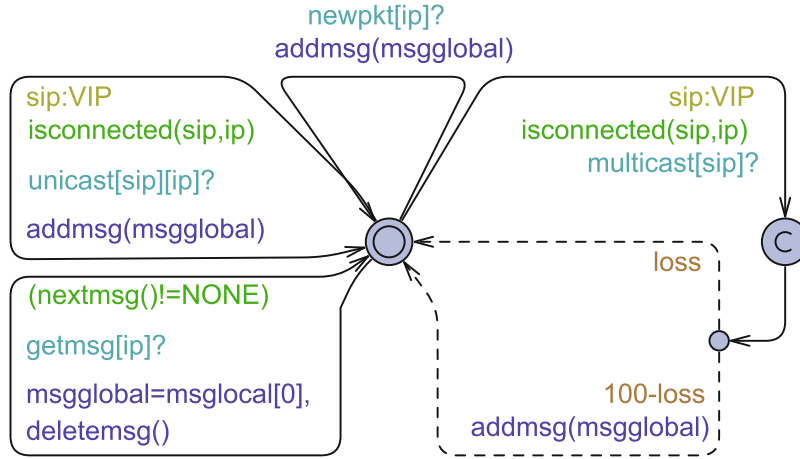


Fig. 4. Queue automaton.

4.3 Node

The model for a node using a reactive protocol comprises of two automata, the *Queue* and the *Handler*, proactive protocols also include a *Generator*. If the protocol uses timers, the model includes a fourth automaton to manage the timers. The generic model defines a number of global variables and channels to facilitate synchronisation between these parts. For instance, channel `img[N]` is an urgent channel which is used for synchronisation between *Handler* and *Queue* automata.

Queue. The template of the queue defines a number of local constants and variables to manage the stored messages. The most important variable is an array `MSG msglocal[QLength]`. The reference model includes methods `void addmsg(MSG msg)` and `void deletemsg()` to add or delete messages from the queue. For synchronisation with the *Handler* the model includes a global variable `bool isMsgInQ[N]` to encode whether a queue contains at least one message.

The automaton for the queue has essentially one control location, as depicted in Fig.4. It has one self-loop for unicast messages, and one loop for multicast messages that can be received, and one for new packets that are inserted by the tester. The latter loop includes a probabilistic choice to lose the message with probability of `loss`. The automaton also includes a loop, labelled `getmsg?`, for the handler to request the first element of the queue.

Handler (Reactive/Proactive). Nodes have routing tables that store information about other nodes of the network which are empty (initialised to 0 at the beginning) and may be updated when they receive control messages from their neighbour nodes (conditions on when to update routing tables can be specific to each protocol). The reference implementation defines an entry to the routing table as:

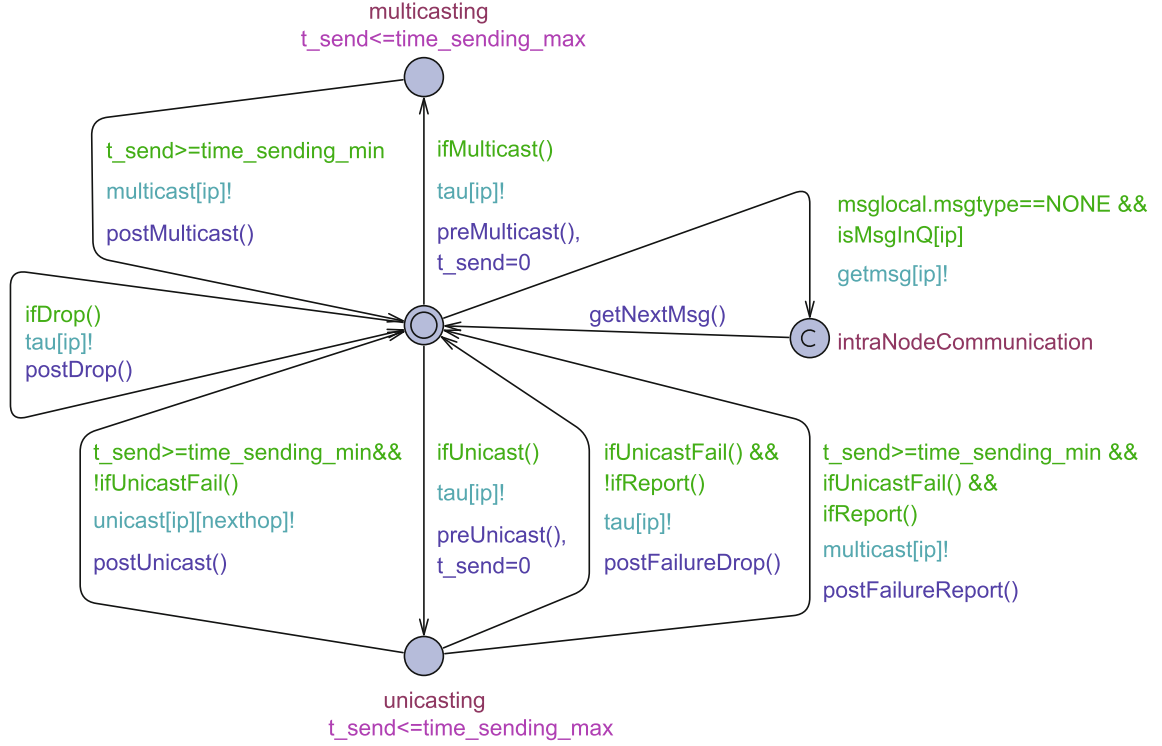


Fig. 5. Handler automaton.

```

typedef struct
{
  IP dip;
  SQN dsn;
  int hops;
  IP nhop;
} rentry;

```

The routing tables are then defined as `rentry art[N][N]`. This is a global variable to allow the *Generator* and verification part read access. Protocols may define additional data structures, for example for error handling.

The handler has a main control location, as depicted in Fig. 5. It includes one loop for multicast messages. The transition to location `multicasting` prepares the message, and waiting in that location up to the permitted amount of time models the message delay, then the transition back to the main location, actually copies the message to the global variable `msgglobal`, for the queue of the receivers to read. The loop for unicast messages has a similar setup, except that includes the option of failure, as in Fig. 3. One option is to drop the message, the other is to multicast an error message. The model also includes for each message type a loop that drops the message, and a loop that requests a new message from the queue.

Most routing protocols use sequence numbers to keep track of newly received information. The reactive version of the handler includes the sequence number, in the proactive version this is a task for the *Generator*.

Generator (Proactive). The task of the *Generator* in a proactive protocol (time-dependent) is to create messages at regular intervals based on the protocol specification. There may be more than one generator, generating more than one type of message. The *Generator* will use a number of clocks to time the generation of those messages as well as to model the sending time of messages (message delays).

Timers (Optional). The protocol may include a fixed number of timers. The model includes as global variables the number of timers, the threshold for each timer, and boolean flags for restart, and to notify that a timer has expired. The automaton managing all timers of one node, has an array of clocks. Once the handler sets the restart flag for a timer to true, the timer automaton will reset a corresponding clock. This transition is urgent, ensuring that no time expires between setting the restart flag, and resetting the corresponding clock. After the threshold duration of the timer, the flag for an expired timer will be set.

4.4 Verification

The model includes for verification purposes a number of side-effect free functions, that can be used by the properties. This includes function to count the number of delivered packets, and how many routes have been established.

For verification the model also includes an automaton *Tester*. The reference model includes an automaton that injects one new packet after about 50 rounds of communication, after which it proceeds to location `final`.

The reference implementation focuses on three main properties, namely for route establishment, network knowledge and packet delivery. These properties verify the core functionality of protocols, e.g., routing data packets.

Route Establishment. The reference model includes the following property for route establishment:

```
Pr[<=1000](<>(route_establishment(OIP1,DIP1)))
```

The function `route_establishment()` returns true if the node `OIP1`, the source node, has the information about the destination node `DIP1` to later send data packet to the destination. The property computes the probability that the function `route_establishment()` returns true in less than or equal to 1000 time units (this value can be altered based on the system requirements).

Network Knowledge. The reference model includes the following property for the network knowledge:

```
E[<=1000;100](max:total_knowledge())
```

This property computes the expected number of connections that have been discovered by time 1000. The function `total_knowledge()` counts for how many originator/destination pairs a route is known.

Packet Delivery. The property for packet delivery is as follows:

```
E[<=1000;100](max:packet_delivered())
```

Packet delivery property shows the number of data packets being delivered at their destinations. The property returns the expected value during 1000 time units by 100 runs. Function `packet_delivered()` is used to count the number of delivered packets.

5 Experiments

We model three well-known routing protocols of WMNs, namely BATMAN, OLSR, AODVv2, to show the reusability and adaptability of our framework. We also verify these protocols for the three different properties discussed in the last section. Each of these is considered for the following scenarios: (1) 0% message loss and no link failures, (2) 80% message loss and no link failures, and (3) 0% message loss and possible link failures.

For all of our experiments, we inject one packet to an arbitrary source to be delivered to an arbitrary destination. We consider networks in a grid, a linear, a fully connected or a ring topology consisting of 9 nodes, as our framework is independent of the number of nodes in the network. It means that number of nodes in the network is adjustable (larger networks are allowed to have) as long as the tool can manage the number of states in the system (state space should be manageable by the tool). A detailed study of these protocols and verification of the models for all possible combinations and/or larger networks are out of the scope of this paper; we only illustrate the applicability and power of the proposed framework.

The automaton `TopologyChanger` is included in models that exhibits link failure. The rate of failure can be simply adjusted based on the protocol specification. We set this value as rate 1 : 10 for all of our models and experiments, e.g., BATMAN, OLSR, AODVv2 models. As none of the protocols that we considered for this study uses timers, we did not have to include the corresponding automaton.

We conduct our experiments using the following set-up: (i) 3.2 GHz Intel Core i5, with 8GB memory, running the Mac OS X 10.11.6 “El Capitan” operating system; (ii) Uppaal SMC model-checker 64-bit version 4.1.20. In Uppaal SMC, two main statistical parameters α and ϵ , in the interval $[0, 1]$, must be fixed by the user. These parameters indicate the probability of *false negatives* and probabilistic *uncertainty*, respectively. In our experiments, these values, i.e., false negatives (α) and probabilistic uncertainty (ϵ) are both set to 0.05, leading to a confidence level of 95%.

5.1 Better Approach to Mobile Ad-hoc Networking (BATMAN)

BATMAN [19] is a proactive protocol used in WMNs. It decentralises route information, i.e., no node has all the data. Each node only maintains information about the possible best next hop. The protocol has two main aims: first, it

discovers all bidirectional links and then identifies the best next hop neighbour for all the other nodes in the network. To provide this information, each node broadcasts originator messages (OGMs) through the network at a regular interval. A node keeps track of the information about other known nodes, stored in the node's routing table. When a node receives a message from its neighbours, it updates this information.

Since BATMAN is a proactive protocol, we include a *Generator* for creating OGMs in addition to the *Handler* and the *Queue*. We adapted our framework based on the model of [4]. Table 1 shows the result of our verification for different topologies. We ran the same experiments for the original model of [4] and we got similar results as we got for our adjusted BATMAN model.

Results show that in case of reliable communication (0% loss), the source nodes can find a path to the destination of the injected packet with the probability in the interval [0.90–1.00] and the routing tables are all populated by periodic exchanging of control messages as they were expected by the specification. The injected packet is delivered at the destination in all types of topologies. When message loss increases to 80%, these values may decrease. The same happens also in case of possible link failures. The verification of the route establishment property (probability estimation) takes on average about 2s whereas the verification for calculating the number of delivered packets and routing table entries (expected value evaluation) takes on average about 215s (calculating the expected values is more time-consuming compared to estimating the probability).

5.2 Optimised Link State Routing (OLSR)

OLSR[5], a proactive protocol used in WMNs, bears the benefit of having routes to different destinations available to be used whenever needed. This is done by exchanging control messages, namely HELLO and Topology Control (TC), periodically through the network. Receiving nodes update their routing tables based on the information in the messages so that when a packet to be destined to some destination is injected, it can find the path in routing tables.

OLSR differs from other proactive protocols in the way that it minimises flooding of control messages by selecting so-called Multipoint Relays (MPRs). Informally, an MPR takes over the communication for a set of nodes that are one-hop neighbours of this node; these one-hop neighbours receive all the routing information from the MRPs and hence do not need to send and receive routing information from other parts of the network.

Our model for each node includes in addition to the routing table also data structures to manage the selection of MPRs. The model includes two *Generators*, one for HELLO and one for TC messages, the *Handler* and the *Queue*. We adapted our framework based on the model of [15] and verified our generic properties. Table 2 shows the result of our verification for different topologies. We ran the same experiments for the original model of [15] and we got similar results as we got for our adjusted OLSR model.

Table 1. BATMAN verification results

	Route establishment			Network knowledge			Packet delivery		
	0% loss	80% loss	link failure	0% loss	80% loss	link failure	0% loss	80% loss	link failure
Grid	0.90 – 1.00	0.00 – 0.10	0.88 – 0.98	72	7	70	1	0	0.4
Linear	0.90 – 1.00	0.00 – 0.10	0.00 – 0.10	72	4	36	1	0	0
Fully connected	0.90 – 1.00	0.25 – 0.35	0.90 – 1.00	72	22	72	1	0.1	0.8
Ring	0.90 – 1.00	0.24 – 0.34	0.85 – 0.95	72	4	58	1	0.1	0.6

Table 2. OLSR verification results

	Route establishment			Network knowledge			Packet delivery		
	0% loss	80% loss	link failure	0% loss	80% loss	link failure	0% loss	80% loss	link failure
Grid	0.90 – 1.00	0.33 – 0.43	0.90 – 1.00	72	65	70	1	0	0.3
Linear	0.90 – 1.00	0.00 – 0.10	0.18 – 0.28	72	34	50	1	0	0
Fully connected	0.90 – 1.00	0.90 – 1.00	0.89 – 0.99	72	72	72	1	1	0.5
Ring	0.90 – 1.00	0.90 – 1.00	0.89 – 0.99	72	43	63	1	1	0.6

Table 3. AODVv2 verification results

	Route establishment			Network knowledge			Packet delivery		
	0% loss	80% loss	link failure	0% loss	80% loss	link failure	0% loss	80% loss	link failure
Grid	0.90 – 1.00	0.00 – 0.10	0.14 – 0.24	28	3	21	1	0	0.3
Linear	0.90 – 1.00	0.00 – 0.10	0.00 – 0.10	72	1	8	1	0.1	0
Fully connected	0.90 – 1.00	0.16 – 0.26	0.90 – 1.00	9	21	11	1	0.1	0.8
Ring	0.90 – 1.00	0.11 – 0.21	0.79 – 0.89	30	2	10	1	0	0.7

Results indicate that in case of reliable communication (0% loss), the source nodes can find a path to the destination of the injected packet with the probability in the interval [0.90–1.00] and the routing tables are all populated by periodic exchanging of control messages as they were expected by the specification. The injected packet is delivered at the destination in all types of topologies. When message loss increases to 80%, these values may decrease. The same happens also in case of possible link failures. The verification regarding route establishment property (probability estimation) takes on average about 2s whereas the verification for calculating the number of delivered packets and routing table entries (expected value evaluation) takes on average about 185s (calculating the expected values is more time-consuming compared to estimating the probability).

5.3 Ad-Hoc On-Demand Distance Vector Version2 (AODVv2)

AODVv2 [21], a reactive protocol for WMNs, behaves on-demand. This means that it tries to find a route to the destination when a packet is injected into the network. The protocol initiates RREQ message and the receiving nodes update their routing tables and possibly rebroadcast the message until the RREQ is received by its destination. Then the destination sends a RREP message back to the source of the RREQ. In this way, a path from the source to the destination is created and the packet can be forwarded via that path. AODVv2 will report failure of links by multicasting RERR messages.

The model of AODVv2 protocol contains only models for the *Handler* and the *Queue*. As a reactive protocol, it does not need a generator. Compared to the other two models, AODVv2 has more message types, as it includes error reporting. This also means that in addition to routing information, each node maintains information of routing errors. We adapted our framework based on the model of [16] and verified our generic properties. Table 3 shows the result of our verification for different topologies. We ran the same experiments for the original model of [16] and we got similar results as we got for our adjusted AODVv2 model which shows the adaptability and reusability of our framework.

Results show that in case of reliable communication (0% loss), the source nodes can find a path to the destination of the injected packet with the probability in the interval [0.90–1.00]. Routing tables are partially populated by periodic exchanging of control messages as they were expected by the specification (reactive protocols find paths to destinations on-demand, so it is expectable that not all tables are updated). However in the linear topology, all routing tables are updated due to the path accumulation feature of AODVv2, meaning that whenever a control message travels via more than one node, information about all intermediate nodes is accumulated in the message and then is distributed to its recipients.

The injected packet is delivered at the destination in all types of topologies in case of reliable communication (0% message loss). When message loss increases to 80%, these values may decrease. The same happens also in case of possible link failures. The verification regarding route establishment property (probability estimation) takes on average about 2s whereas the verification for calculating the number of delivered packets and routing table entries (expected value evaluation) takes on average about 15s (calculating the expected values is more time-consuming compared to estimating the probability).

Evaluating the expected values for AODVv2 takes less time due to the reactive characteristic of AODVv2, meaning that since AODVv2 broadcasts control messages on demand it has less number of states compared to BATMAN and OLSR that broadcast control messages periodically which decreases the time spent for verification. As all the three protocols are modelled applying our framework, it is possible to easily compare the protocols w.r.t. the properties and verification time.

5.4 Discussion on BATMAN, OLSR and AODVv2 Models

Here, we discuss how much our framework needs the interaction from the modeller to be adjusted based on the protocol specification. In other words, how much the three case studies and our framework have in common and how much they are different. The general structure of our six automata (locations and transitions of the automata), i.e., *Handler*, *Queue*, *Generator*, *Timer*, *Topology-Changer* and *Tester*, and their synchronisation remain unchanged and only some declaration (code fragments) of the automata may need to be modified/added based on the specification of the protocol.

- Communication: format of each message has been separately modified using *typedef struct* (based on BATMAN, OLSR and AODVv2 specifications) and later is added in our generic message *MSG*. The *IP* address of nodes, *SEQN* sequence numbers, channels, etc are borrowed from the framework.
- Topology: connectivity function, network topology and *TopologyChanger* automata remain unchanged. We have only borrowed them from our framework.
- Node: the *Queue* and the *Timer* automata remain unchanged and they have been only imported and used. Function *createMSG* in the *Generator* declaration which is applicable only for BATMAN and OLSR, has been modified based on the specification (as mentioned earlier, the format of messages for different protocols are unique to the protocol and must be changed based on the specification). The interval for sending periodic messages is a parameter of each protocol and should be set in the declarations.

The *Handler* needs more interactions from the modeller when modifying the local declaration of the automaton. This is the case due to different behaviour of protocols, e.g., when to update a routing table, when to process/drop a message, when to multicast a message, etc. For instance, BATMAN protocol has a specific procedure for storing sequence numbers which is unique to this protocol, OLSR has a specific procedure for determining MPRs, and AODVv2 has a specific procedure for accumulating paths. These specific features need to be separately modelled for each protocol and our framework only supports the standardise behaviour of routing protocols which were discussed earlier. Our models move much of the logic to functions inside the model in order to have the core of the protocol as code fragments in the model which makes the modelling task easier.

- Verification: the three system requirements (properties) and their corresponding functions have also been imported without any modifications. The *Tester* automaton injects the packet in accordance to the category of the protocol; reactive or proactive protocol. If the protocol is proactive (BATMAN and OLSR), the *Tester* automaton injects the packet after routes are discovered; and if the protocol is reactive (AODVv2), the *Tester* injects the packet for route discovery process and the routes are discovered later after packet injection. It means that only the time interval that the *Tester* transition is enabled differs for reactive and proactive protocols.

6 Related Work

Formal modelling and analysis of the WMNs and Mobile Ad-hoc Networks (MANETs) and their routing protocols is among challenging tasks, and formal verification of such systems has attracted the attention from formal methods community [3, 11, 17]. Fehnker et al. [8] applied the Uppaal model checker [2] for analysing qualitative properties of the AODV protocol in all network topologies with five nodes. Kamali et al. [15] focused on formal modelling and verifying OLSR protocol in network topologies with five nodes. They have also applied

Event-B to model OLSR and have analysed this protocol in large networks (no size barrier w.r.t. the size of the network) [14]. Chaudhary et al. [4] formally modelled BATMAN routing protocol using Uppaal model checker revealing several ambiguities in the RFC. They verified their model for loop-freedom, bidirectional link discovery, and route-discovery. Fehnker et al. [9] modelled and verified LMAC protocol of wireless sensor networks applying Uppaal. Their study was carried out to detect and resolve collision in networks consisting of four and five nodes.

There are several studies using (statistical) model checking to analyse WMN and MANET routing protocols. Höfner and McIver [13] made a comparison of the AODV and DYMO protocols on arbitrary networks up to five nodes considering perfect communication among nodes, applying the Uppaal SMC model checker. Their analysis shows that DYMO has worsened performance compared to AODV. Dal Corso et al. [6] studied the extended and generalised work done by [13] to 4×3 grids with lossy communication. They showed contrary results, indicating that DYMO is performing better compared to AODV. Kamali et al. [16] investigated and compared the performance and looping property of the most recent version of AODV protocol [21] with DYMO on 3×3 grids. Their results indicate that the more recent version of AODV pays the price of degraded performance compared to DYMO to remain loop-free.

There are other studies providing formal frameworks for modelling and verifying MANETs. Liu et al. [18] presented a formal modelling framework for MANETs consisting of several mobility models together with wireless communication applying Real-Time Maude [20]. They analysed the AODV protocol using their framework and their mobility models. Their framework mainly focuses only on integrating a number of mobility models together with wireless communication. Yousefi et al. [22] have modelled MANETs using the extension of an actor-based modelling language bRebeca. They provided a framework to detect malfunctioning of MANET protocols, addressing local broadcast and topology changes. They have modelled the core functionality of AODV protocol and found some malfunctioning of this protocol (loop existence).

Our work differs from the other previous works in the sense that it models the core functionality of WMN routing protocols, considering wireless communication, topology, message loss, message queuing, link failure, etc. It is also possible to model timing aspects of protocols (both reactive and proactive) and to allow probabilities to have both qualitative and quantitative analysis.

In addition, networks of timed automata as the specification language used for introducing our generic framework (the main common components of routing protocols) are more manageable to alter based on the protocols specifications. It means that adapting our framework allows protocol designers to have an insight of the system before the deployment since timed automata is an easy-to-understand specification language and Uppaal SMC simulator provides the means to validate the system which later can be also used for verification. Protocol designers can simply modify the C-like code in the declarations based on

the protocol specification where the general structure of networks of different automata remains unchanged.

7 Conclusion

This paper presented an adaptive, generic and reusable framework as well as crucial generic properties to model and verify WMN routing protocols. This framework uses Uppaal SMC to capture timing aspect of protocols, communication between nodes, and probabilities to model message loss, link breakage, etc.

This paper discussed the general structure of Uppaal models, and how this influences the design of models for network routing protocols. It described how to build reusable components within the constraints imposed by the Uppaal modelling language. It identified the main components that routing protocols have in common, and how to map them to data structures, processes, channels, and timed automata in the Uppaal language. We demonstrated the applicability of the approach by implementing three different protocols in this framework: AODVv2, OLSR and BATMAN.

One of the characteristics of these models is that they move much of the logic to functions inside of the Uppaal model. They rely less on the subtle interplay of channels, urgent locations, or committed locations. Instead, they standardise proven patterns that have been used in the community to model routing protocols. This also means that the core of the protocol resides as code fragments in the model, and becomes available to be standard code reviewing practices.

An observation that was made is that Uppaal as modelling language would benefit if it would adopt more mechanisms to structure code. It would be beneficial if the model could reflect that a number of templates share access to data structures to the exclusion of others. Often the workaround for sharing information is to make data global, without mechanisms to enforce its consistent use. Furthermore, code that is included for verification is currently scattered across the model. It might be worth to consider verification as a cross-cutting concern, similarly to how these are dealt with in aspect-oriented programming.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. *Theor. Comput. Sci.* **126**(2), 183–235 (1994)
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) *SFM-RT 2004*. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
3. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. *J. ACM* **49**(4), 538–576 (2002)
4. Chaudhary, K., Fehnker, A., Mehta, V.: Modelling, verification, and comparative performance analysis of the B.A.T.M.A.N. protocol. In: Hermanns, H., Höfner, P. (eds.) *MARS 2017*, vol. 244, pp. 53–65 (2017)

5. Clausen, T., Jacquet, P.: Optimized link state routing protocol (OLSR). RFC3626 (2003). <http://www.ietf.org/rfc/rfc3626>
6. Dal Corso, A., Macedonio, D., Merro, M.: Statistical model checking of ad hoc routing protocols in lossy grid networks. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 112–126. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17524-9_9
7. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal SMC tutorial. STTT **17**(4), 397–415 (2015)
8. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 173–187. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_13
9. Fehnker, A., van Hoesel, L., Mader, A.: Modelling and verification of the LMAC protocol for wireless sensor networks. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 253–272. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73210-5_14
10. Fehnker, A., Höfner, P., Kamali, M., Mehta, V.: Topology-based mobility models for wireless networks. In: Joshi, K., Siegle, M., Stoelinga, M., D’Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 389–404. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40196-1_32
11. van Glabbeek, R., Höfner, P., Portmann, M., Tan, W.L.: Modelling and verifying the AODV routing protocol. Distrib. Comput. **29**(4), 279–315 (2016)
12. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) Verification, Model Checking, and Abstract Interpretation, pp. 73–84. Springer, Berlin (2004)
13. Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 322–336. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_22
14. Kamali, M., Petre, L.: Modelling link state routing in event-B. In: Wang, H., Mokhtari, M. (eds.) ICECCS 2016, pp. 207–210. IEEE (2016)
15. Kamali, M., Höfner, P., Kamali, M., Petre, L.: Formal analysis of proactive, distributed routing. In: Calinescu, R., Rumpe, B. (eds.) SEFM 2015. LNCS, vol. 9276, pp. 175–189. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22969-0_13
16. Kamali, M., Merro, M., Dal Corso, A.: AODVv2: performance vs. loop freedom. In: Tjoa, A.M., Bellatreche, L., Biffl, S., van Leeuwen, J., Wiedermann, J. (eds.) SOFSEM 2018. LNCS, vol. 10706, pp. 337–350. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73117-9_24
17. Kamali, M., Petre, L.: Improved recovery for proactive, distributed routing. In: ICECCS 2015, pp. 178–181. IEEE (2015)
18. Liu, S., Ölveczky, P.C., Meseguer, J.: A framework for mobile ad hoc networks in real-time maude. In: Escobar, S. (ed.) WRLA 2014. LNCS, vol. 8663, pp. 162–177. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12904-4_9
19. Neumann, A., Aichele, C., Lindner, M., Wunderlich, S.: Better approach to mobile ad-hoc networking (BATMAN). Internet draft00 (2008). <https://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00>
20. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. High.-Order Symb. Comput. **20**(1), 161–196 (2007)

21. Perkins, C., Stan, R., Dowdell, J., Steenbrink, L., Mercieca, V.: Ad hoc on-demand distance vector version 2 (AODVv2) routing. Internet Draft 16 (2016). <https://datatracker.ietf.org/doc/draft-ietf-manet-aodvv2>
22. Yousefi, B., Ghassemi, F., Khosravi, R.: Modeling and efficient verification of wireless ad hoc networks. *Form. Asp. Comput.* **29**(6), 1051–1086 (2017)

Turku Centre for Computer Science

TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4 -Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen**, On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Sääntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Methods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performance Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages
159. **Peter Sarlin**, Mapping Financial Stability
160. **Alexander Wei Yin**, On Energy Efficient Computing Platforms
161. **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
162. **Maryam Kamali**, Reusable Formal Architectures for Networked Systems
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

167. **Bo Yang**, Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms
168. **Ali Hanzala Khan**, Consistency of UML Based Designs Using Ontology Reasoners
169. **Sonja Leskinen**, m-Equine: IS Support for the Horse Industry
170. **Fareed Ahmed Jekhio**, Video Transcoding in a Distributed Cloud Computing Environment
171. **Moazzam Fareed Niazi**, A Model-Based Development and Verification Framework for Distributed System-on-Chip Architecture
172. **Mari Huova**, Combinatorics on Words: New Aspects on Avoidability, Defect Effect, Equations and Palindromes
173. **Ville Timonen**, Scalable Algorithms for Height Field Illumination
174. **Henri Korvela**, Virtual Communities – A Virtual Treasure Trove for End-User Developers
175. **Kameswar Rao Vaddina**, Thermal-Aware Networked Many-Core Systems
176. **Janne Lahtiranta**, New and Emerging Challenges of the ICT-Mediated Health and Well-Being Services
177. **Irum Rauf**, Design and Validation of Stateful Composite RESTful Web Services
178. **Jari Björne**, Biomedical Event Extraction with Machine Learning
179. **Katri Haverinen**, Natural Language Processing Resources for Finnish: Corpus Development in the General and Clinical Domains
180. **Ville Salo**, Subshifts with Simple Cellular Automata
181. **Johan Ersfolk**, Scheduling Dynamic Dataflow Graphs
182. **Hongyan Liu**, On Advancing Business Intelligence in the Electricity Retail Market
183. **Adnan Ashraf**, Cost-Efficient Virtual Machine Management: Provisioning, Admission Control, and Consolidation
184. **Muhammad Nazrul Islam**, Design and Evaluation of Web Interface Signs to Improve Web Usability: A Semiotic Framework
185. **Johannes Tuikkala**, Algorithmic Techniques in Gene Expression Processing: From Imputation to Visualization
186. **Natalia Díaz Rodríguez**, Semantic and Fuzzy Modelling for Human Behaviour Recognition in Smart Spaces. A Case Study on Ambient Assisted Living
187. **Mikko Pänkäälä**, Potential and Challenges of Analog Reconfigurable Computation in Modern and Future CMOS
188. **Sami Hyrynsalmi**, Letters from the War of Ecosystems – An Analysis of Independent Software Vendors in Mobile Application Marketplaces
189. **Seppo Pulkkinen**, Efficient Optimization Algorithms for Nonlinear Data Analysis
190. **Sami Pyötiälä**, Optimization and Measuring Techniques for Collect-and-Place Machines in Printed Circuit Board Industry
191. **Syed Mohammad Asad Hassan Jafri**, Virtual Runtime Application Partitions for Resource Management in Massively Parallel Architectures
192. **Toni Ernvall**, On Distributed Storage Codes
193. **Yuliya Prokhorova**, Rigorous Development of Safety-Critical Systems
194. **Olli Lahdenoja**, Local Binary Patterns in Focal-Plane Processing – Analysis and Applications
195. **Annika H. Holmbom**, Visual Analytics for Behavioral and Niche Market Segmentation
196. **Sergey Ostroumov**, Agent-Based Management System for Many-Core Platforms: Rigorous Design and Efficient Implementation
197. **Espen Suenson**, How Computer Programmers Work – Understanding Software Development in Practise
198. **Tuomas Poikela**, Readout Architectures for Hybrid Pixel Detector Readout Chips
199. **Bogdan Iancu**, Quantitative Refinement of Reaction-Based Biomodels
200. **Ilkka Törmä**, Structural and Computational Existence Results for Multidimensional Subshifts
201. **Sebastian Okser**, Scalable Feature Selection Applications for Genome-Wide Association Studies of Complex Diseases
202. **Fredrik Abbors**, Model-Based Testing of Software Systems: Functionality and Performance
203. **Inna Pereverzeva**, Formal Development of Resilient Distributed Systems
204. **Mikhail Barash**, Defining Contexts in Context-Free Grammars
205. **Sepinoud Azimi**, Computational Models for and from Biology: Simple Gene Assembly and Reaction Systems
206. **Petter Sandvik**, Formal Modelling for Digital Media Distribution

207. **Jongyun Moon**, Hydrogen Sensor Application of Anodic Titanium Oxide Nanostructures
208. **Simon Holmbacka**, Energy Aware Software for Many-Core Systems
209. **Charalampos Zinoviadis**, Hierarchy and Expansiveness in Two-Dimensional Subshifts of Finite Type
210. **Mika Murtojärvi**, Efficient Algorithms for Coastal Geographic Problems
211. **Sami Mäkelä**, Cohesion Metrics for Improving Software Quality
212. **Eyal Eshet**, Examining Human-Centered Design Practice in the Mobile Apps Era
213. **Jetro Vesti**, Rich Words and Balanced Words
214. **Jarkko Peltomäki**, Privileged Words and Sturmian Words
215. **Fahimeh Farahnakian**, Energy and Performance Management of Virtual Machines: Provisioning, Placement and Consolidation
216. **Diana-Elena Gratie**, Refinement of Biomodels Using Petri Nets
217. **Harri Merisaari**, Algorithmic Analysis Techniques for Molecular Imaging
218. **Stefan Grönroos**, Efficient and Low-Cost Software Defined Radio on Commodity Hardware
219. **Noora Nieminen**, Garbling Schemes and Applications
220. **Ville Taajamaa**, O-CDIO: Engineering Education Framework with Embedded Design Thinking Methods
221. **Johannes Holvitie**, Technical Debt in Software Development – Examining Premises and Overcoming Implementation for Efficient Management
222. **Tewodros Deneke**, Proactive Management of Video Transcoding Services
223. **Kashif Javed**, Model-Driven Development and Verification of Fault Tolerant Systems
224. **Pekka Naula**, Sparse Predictive Modeling – A Cost-Effective Perspective
225. **Antti Hakkala**, On Security and Privacy for Networked Information Society – Observations and Solutions for Security Engineering and Trust Building in Advanced Societal Processes
226. **Anne-Maarit Majanoja**, Selective Outsourcing in Global IT Services – Operational Level Challenges and Opportunities
227. **Samuel Rönqvist**, Knowledge-Lean Text Mining
228. **Mohammad-Hashem Hahgbayan**, Energy-Efficient and Reliable Computing in Dark Silicon Era
229. **Charmi Panchal**, Qualitative Methods for Modeling Biochemical Systems and Datasets: The Logiscope and the Reaction Systems Approaches
230. **Erkki Kaila**, Utilizing Educational Technology in Computer Science and Programming Courses: Theory and Practice
231. **Fredrik Robertsén**, The Lattice Boltzmann Method, a Petaflop and Beyond
232. **Jonne Pohjankukka**, Machine Learning Approaches for Natural Resource Data
233. **Paavo Nevalainen**, Geometric Data Understanding: Deriving Case-Specific Features
234. **Michal Szabados**, An Algebraic Approach to Nivat’s Conjecture
235. **Tuan Nguyen Gia**, Design for Energy-Efficient and Reliable Fog-Assisted Healthcare IoT Systems
236. **Anil Kanduri**, Adaptive Knobs for Resource Efficient Computing
237. **Veronika Suni**, Computational Methods and Tools for Protein Phosphorylation Analysis
238. **Behailu Negash**, Interoperating Networked Embedded Systems to Compose the Web of Things
239. **Kalle Rindell**, Development of Secure Software: Rationale, Standards and Practices
240. **Jurka Rahikkala**, On Top Management Support for Software Cost Estimation
241. **Markus A. Whiteland**, On the k-Abelian Equivalence Relation of Finite Words
242. **Mojgan Kamali**, Formal Analysis of Network Routing Protocols

TURKU CENTRE *for* COMPUTER SCIENCE

<http://www.tucs.fi>
tucs@abo.fi



University of Turku

Faculty of Science and Engineering

- Department of Future Technologies
- Department of Mathematics and Statistics

Turku School of Economics

- Institute of Information Systems Science



Åbo Akademi University

Faculty of Science and Engineering

- Computer Engineering
- Computer Science

Faculty of Social Sciences, Business and Economics

- Information Systems

ISBN 978-952-12-3844-4
ISSN 1239-1883

Mojgan Kamali

Mojgan Kamali

Mojgan Kamali

Formal Analysis of Network Routing Protocols

Formal Analysis of Network Routing Protocols

Formal Analysis of Network Routing Protocols