Formal Analysis of Proactive, Distributed Routing

Mojgan Kamali[1], Peter Höfner[2,3], Maryam Kamali[4], and Luigia Petre[1]

[1] Åbo Akademi University, Finland
[2] NICTA, Australia
[3] University of New South Wales, Australia
[4] University of 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.[5] 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 Section 2, we overview the OLSR protocol and in Section 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 Section 5 and propose future research directions in Section 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

[5] 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 MRPs 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 2 seconds 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 5 seconds 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 6 seconds (three times the interval between sending HELLO messages); information from TC messages for 15 seconds (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**. $\mathbf{A}\phi$ means that the formula ϕ holds for all paths starting from the current state. $\mathbf{G}\phi$ means all future states (including the current one) satisfy ϕ. Formulas combine the path quantifies and the temporal operators, e.g. $\mathbf{AG}\phi$ 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].

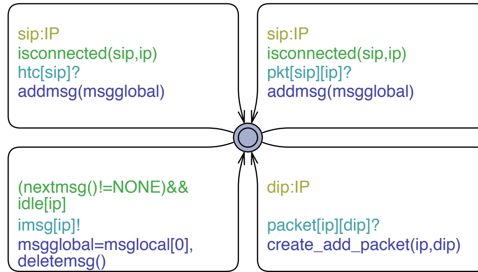

Fig. 1: The `Queue` automaton.

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
  } rtentry;
```

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 milliseconds. Considering a sending time of 500 milliseconds (in our model time_sending = 500), nodes have to broadcast a new message 1500 milliseconds 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, 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 milliseconds.

After MPR nodes have been selected, each of them prepares for broadcasting TC messages to the connected nodes. TC messages are sent every 5000 milliseconds. 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 ex-

ample, 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].

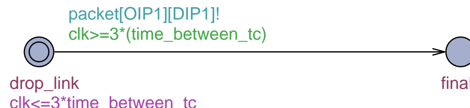

Fig. 2: The Tester1 automaton.

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

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

Remember that the CTL formula $\texttt{A[]}\phi$ 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

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

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.[6] Property 3 is expressed as

$$\texttt{A[]}\ ((\texttt{Tester1.final}\ \&\&\ \texttt{node(OIP1).a} \mathrel{!}= 0)\ \texttt{imply}\\ \texttt{node(OIP1).rt[DIP1].hops} == 3) \tag{3}$$

Here, $\texttt{node(OIP1).a}! = 0$ indicates whether OIP1 has sent its packet to the next node along the path to DIP1; node(OIP1).rt[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 minutes.

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 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

[6] There are too many topologies of that size, so we cannot analyse all topologies.

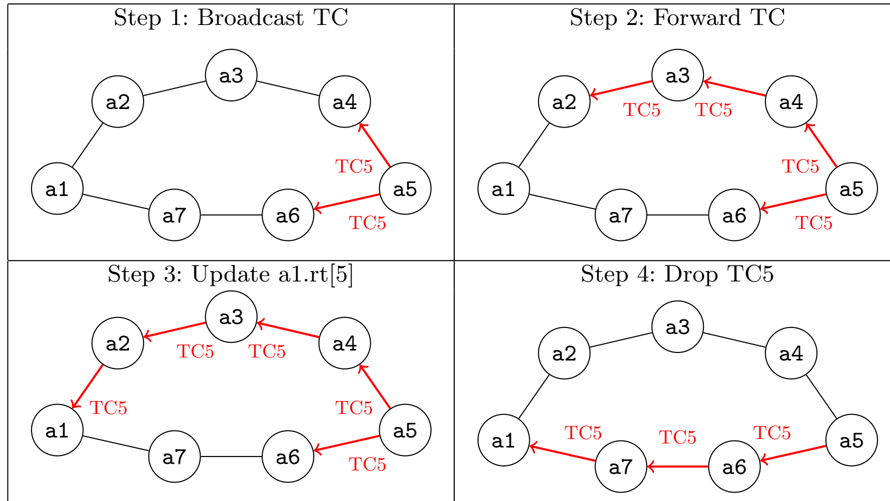

Table 1: Establishment of non-optimal routes in a 7 node topology

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*(time_between_tc + time_sending)` (in our model at

15000 milliseconds), 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.

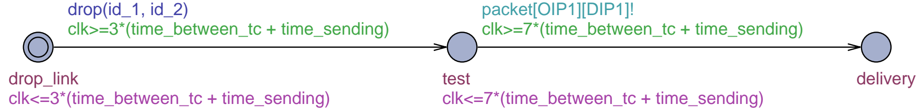

Fig. 3: The `Tester2` automaton.

Based on RFC 3626 (see box below), the information about one-hop and two-hop neighbours of a node is valid for `3*REFRESH_INTERVAL`, which equals 6000 milliseconds; information about nodes which are more than two hops away from that node is valid for `3*TC_INTERVAL`, that equals 15000 miliseconds.

> "NEIGHB_HOLD_TIME = 3*REFRESH_INTERVAL
> TOP_HOLD_TIME = 3*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 milliseconds; this indicates the breakage of a link. Also, if a node has not received TC messages from other MPR nodes for more than 15 seconds, 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

$$\texttt{A[]}\ ((\texttt{Tester2.delivery}\ \&\&\ \texttt{emptybuffers}())\ \texttt{imply}\ \texttt{node}(\texttt{DIP1}).\texttt{delivered} \mathrel{!=} \texttt{0}) \tag{4}$$

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. ($\texttt{node}(\texttt{DIP1}).\texttt{delivered}! = \texttt{0}$).

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 seconds (between 15000–35000 milliseconds), which is a long period; in particular

since we only consider networks of small size. As a consequence, this means that only after 35 seconds, 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 seconds, 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.

$$\texttt{A[]}\ ((\texttt{Tester3.delivery}\ \&\&\ \texttt{emptybuffers}())\ \texttt{imply}\\ \texttt{node}(\texttt{DIP1}).\texttt{delivered} != \texttt{0}) \tag{5}$$

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 milliseconds, the packet is definitely delivered at the destination. Therefore, it is feasible to reduce the recovery time of OLSR about 10000 milliseconds (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 reveales 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: International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004. Revised Lectures. pp. 200–236. Springer Verlag (2004)
2. Chiyangwa, S., Kwiatkowska, M.Z.: A timing analysis of AODV. In: FMOODS. Lecture Notes in Computer Science, vol. 3535, pp. 306–321. Springer (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: 4th International Conference on

Formal Modelling and Analysis of Timed Systems (FORMATS06). Lecture Notes in Computer Science, vol. 4202, pp. 128–142. Springer Berlin Heidelberg (2006)
6. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics, pp. 995–1072. MIT (1995)
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. pp. 1–6. Vancouver (2011)
8. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012). pp. 173–187. Springer, Tallinn, Estonia (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'13). pp. 91–100. ACM (2013)
10. Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: 5th NASA Formal Methods Symposium (NFM 2013). vol. 7871, pp. 322–336. Springer, Moffett Field, CA, USA (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. Tech. Rep. 1125, TUCS – Turku Centre for Computer Science (2014)
14. Kamali, M., Laibinis, L., Petre, L., Sere, K.: Formal development of wireless sensor-actor networks. Science of Computer Programming 80, Part A(0), 25 – 49 (2014)
15. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (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 Systems 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 '10). 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)