AODVv2-16: performance vs. loop freedom

Mojgan Kamali[1], Massimo Merro[2], and Alice Dal Corso[2]

[1] Faculty of Science and Engineering, Åbo Akademi University, Finland
[2] Dipartimento di Informatica, Università degli Studi di Verona, Italy

Abstract. We compare two evolutions of 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 3x3 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 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) [20] protocol to investigate their performance and to analyse if they may yield routing loops. AODV is one of the four protocols standardised by the IETF MANET working group. 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*).

Different studies of 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 the systems for 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) [24] 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 [21] 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 both AODV and DYMO suffer from *routing loops* [5, 10, 14, 19], 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 strongly motivated by a recent version of the AODVv2-16 Internet draft [23], 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 3x3 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 w.r.t. 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 w.r.t. 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 helps to reduce/remove loops.

Outline. In Sec. 2, we overview both DYMO and AODVv2-16. In Sec. 3, we briefly discuss the Uppaal models of the two protocols based on their RFCs [21, 23]. In Sec. 4 and 5, we present the results of our analysis w.r.t. performance and loop occurrences. In Sec. 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 [21] 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 [23] 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 `rrep`s 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 protocol[3].

2.1 Degrading performance to avoid routing loops

Different studies have proved the presence of loops in both AODV and DYMO protocol [5, 10, 14, 19]. Here, we report a simple example to show how a loop can occur in DYMO, and how this is avoided in AODVv2-16.

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

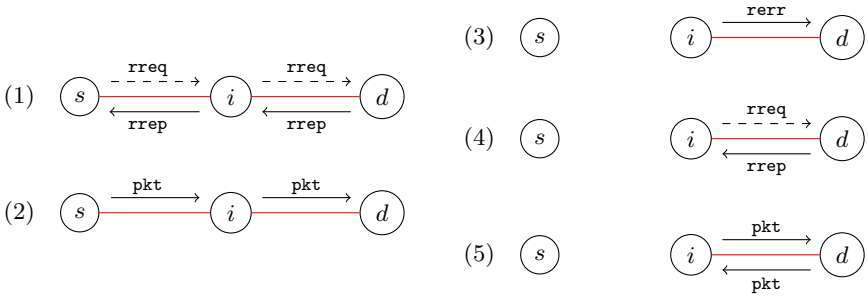


Fig. 1. Presence of a loop in DYMO.

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." [[23], page 28]

Here, `LocalRoutes` stores previously received messages, `AdvRte` contains the information about newly received message, and `LoopFree(AdvRte, LocalRoute):= (Cost(AdvRte) <= Cost(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 DYMO[4]. 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 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 $0 \leq \mathsf{loss} \leq 100$, and a node can either lose a message with weight loss or receive it with weight $(100-\mathsf{loss})$.

[4] The reader can consult our models at http://users.abo.fi/mokamali/SOFSEM2018.

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 [21, 23]. 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, `rreq`s 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 `rrep`s and also conditions regarding updates of the routing tables. As we explained in Section 2, AODVv2-16 tries to find the whole path through the destination node and it does not rely on the `rrep`s 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 w.r.t. 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 40ms), 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 3x3 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 16GB memory, running the Mac OS X 10.9 "Maverick" operating system; (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 3x3 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[OIP1][DIP1].nhop!=0  &&  art[OIP2][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

	loss=0%	st. dev.	loss=10%	s. 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

Table 1. Route establishment on 3x3 grid networks ($\alpha = \epsilon = 0.01$).

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 resent 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.

We can see that on the 3x3 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[<=10000,26492](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

	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

Table 2. Route quantity on 3x3 grid networks (26492 runs for each experiment).

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.

We repeat the same analysis of [15] on our 3x3 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 3x3 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, 18], all ad-hoc routing protocols based on `rreq`-broadcast can establish non-optimal routes when, for instance, the destination node does not forward the `rreq`-message. This phenomenon is more evident in a scenario with unreliable communication.

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

```
Pr[<=10000](<>(tester.final  &&  emptybuffers() &&
art[OIP1][DIP1].hops==min_path  &&  art[OIP2][DIP2].hops==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 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

	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. Optimal routing on 3x3 grid network. ($\alpha = \epsilon = 0.01$).

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 3x3 grids during the routing process, under the assumption that links between nodes can break with a *high*

	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

Table 4. Packet delivery on 3x3 grid networks ($\alpha = \epsilon = 0.01$).

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 the sender node 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.

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 rate 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 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

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

Table 5. Number of loops in different configurations.

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

Table 6. Number of configurations that have loops.

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 [23] 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 3x3 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 the preceding 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 attentions from formal methods community [1, 3, 4, 7, 13, 15, 17, 19]. 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 systematically basic qualitative properties of the AODV routing protocol in all network topologies up to five nodes. Höfner and McIver [15] compare AODV and DYMO on arbitrary networks up to 5 nodes with perfect communication, relying on the Uppaal SMC model checker (release 4.1.11). Dal Corso et al. [7] extends and generalises the work of [15] to 4x3 grids with lossy communication.

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 et al. [19] have investigated the looping property of DYMO 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. [25] have applied their extension of actor-based modelling language bRebeca to model AODVv2-11 [22] (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).

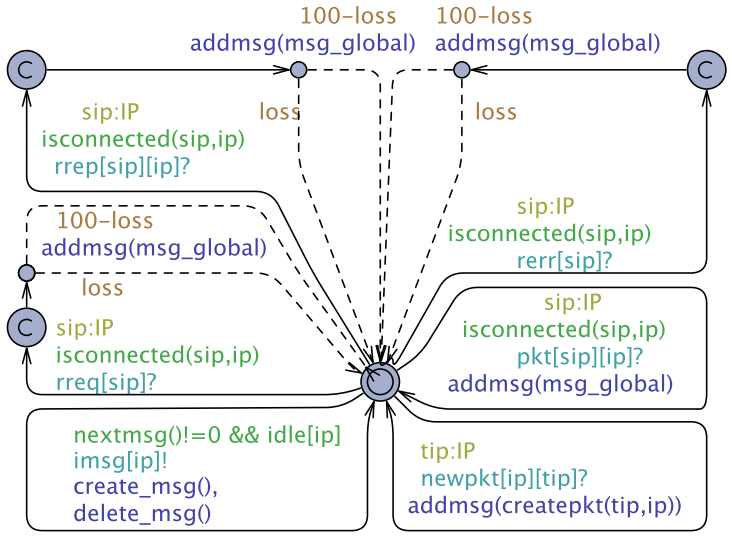
References

1. Battisti, L., Macedonio, D., Merro, M.: Statistical Model Checking of a Clock Synchronization Protocol for Sensor Networks. In: FSEN 2013. LNCS, vol. 8161, pp. 168–182. Springer (2013)
2. 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 (2004)
3. Benetti, D., Merro, M., Viganò, L.: Model Checking Ad Hoc Network Routing Protocols: ARAN vs. endairA. In: SEFM 2010. pp. 191–202. IEEE Computer Society (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., Glabbeek, R., Höfner, P.: A timed process algebra for wireless networks with an application in routing. In: Proceedings of the 25th European Symposium on Programming Languages and Systems - Volume 9632. pp. 95–122 (2016)
6. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
7. Dal Corso, A., Macedonio, D., Merro, M.: Statistical model checking of ad hoc routing protocols in lossy grid networks. In: NASA Formal Methods - 7th International Symposium, NFM. pp. 112–126 (2015)
8. David, A., Larsen, K.G., Legay, A., Mikuăionis, 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: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 7214, pp. 173–187. Springer (2012)
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: 2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems. pp. 426–435 (2004)
13. van Glabbeek, R., Höfner, P., Portmann, M., Tan, W.L.: Modelling and verifying the aodv routing protocol. Distributed Computing 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: Proceedings of the 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'13). pp. 91–100 (2013)
15. Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: NASA Formal Methods Symposium (NFM'13). pp. 322–336 (2013)
16. Kamali, M., Merro, M., Dal Corso, A.: Aodvv2: performance vs. loop freedom. Tech. rep., TUCS – Turku Centre for Computer Science (2016)
17. Kamali, M., Höfner, P., Kamali, M., Petre, L.: Formal analysis of proactive, distributed routing. In: 13th International Conference on Software Engineering and Formal Methods (SEFM 2015). pp. 175–189 (2015)
18. Miskovic, S., Knightly, E.W.: Routing primitives for wireless mesh networks: Design, analysis and experiments. In: INFOCOM, 2010 Proceedings IEEE. pp. 1–9 (2010)

19. Namjoshi, K.S., Trefler, R.J.: Loop freedom in aodvv2. In: Formal Techniques for Distributed Objects, Components, and Systems - 35th IFIP WG 6.1 International Conference, FORTE 2015. pp. 98–112 (2015)
20. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental) (2003)
21. Perkins, C., Stan, R., Dowdell, J.: Dynamic MANET On-demand (AODVv2) Routing draft-ietf-manet-dymo. Internet Draft 26 (2013)
22. 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)
23. Perkins, C., Stan, R., Dowdell, J., Steenbrink, L., Mercieca, V.: Dynamic MANET On-demand (AODVv2) Routing draft-ietf-manet-aodvv2. Internet Draft 16 (2016)
24. Sen, K., Viswanathan, M., Agha, G.A.: Vesta: A statistical model-checker and analyzer for probabilistic systems. In: QEST. pp. 251–252 (2005)
25. Yousefi, B., Ghassemi, F., Khosravi, R.: Modeling and efficient verification of wireless ad hoc networks. CoRR abs/1604.07179 (2016)

This appendix is for refereeing only; it depicts the timed automata `Queue` for reliable and unreliable communication as well as for link breakage.

A `Queue` Automaton in presence of reliable and unreliable communication



B `Queue` Automaton in presence of link breakage