Statistical Model Checking of Ad Hoc Routing Protocols in Lossy Grid Networks*

Alice Dal Corso[1], Damiano Macedonio[2], and Massimo Merro[1]

[1] Dipartimento di Informatica, Università degli Studi di Verona, Italy
[2] Julia Srl, Verona, Italy

Abstract We extend recent work by Höfner and McIver con the performances of the *ad hoc routing protocols* AODV and DYMO in terms of routes established. Höfner and McIver apply *statistical model checking* to show that on arbitrary small networks (up to 5 nodes) the most recent, and apparently more robust, DYMO protocol is less efficient than AODV. Here, we reformulate their experiments on 4x3 toroidal networks, with possibly lossy communication. As a main result we demonstrate that, in this more realistic scenario, DYMO performs significantly better than AODV.

1 Introduction

Ad hoc networking is a relatively recent area in wireless communications that is attracting the attention of many researchers for its potential to provide ubiquitous connectivity without the assistance of any fixed infrastructure. A *Mobile Ad Hoc Network* (MANET) is an autonomous system composed of *mobile* devices communicating with each other via radio transceivers.

Wireless devices use radio frequency channels to *broadcast* messages to the other devices. A single transmission span over a limited area and reach only a subset of the devices in the network. As a consequence, ad hoc networks rely on multi-hop wireless communications where nodes have essentially two roles: (i) acting as end-systems and (ii) performing routing functions.

A *routing protocol* is used to determine the appropriate paths on which data should be transmitted in a network. Routing protocols for wireless systems can be classified into topology-based and position-based ones:

- *Topology-based protocols* rely on traditional routing concepts, such as maintaining routing tables or distributing link-state information.
- *Position-based protocols* use information about the physical locations of the nodes to route data packets to their destinations.

Topology-based protocols can be further divided into proactive protocols and reactive ones:

* The third author is partly supported by the Joint Project 2011 "Statical Analysis for Multithreading" from Università degli Studi di Verona.

- *Proactive routing protocols* try to maintain consistent routing information within the system at any time.
- *Reactive routing protocols* establish a route between a source and a destination only when it is needed, typically when a new data packet is injected by a user. For this reason, reactive protocols are also called *on-demand* protocols.

Examples of proactive routing protocols for mobile ad hoc networks are OLSR [6] and DSDV [19], while DSR [13], AODV [17] and DYMO [18] are typical on-demand protocols.

Most of the analyses of protocols for large-scale MANETs are usually based on discrete-event simulators (e.g., ns-2, Opnet and Glomosim). However, different simulators often support different models of the MAC physical-layer yielding different results, even for simple systems. Formal analysis techniques allow to screen protocols for flaws and to exhibit counterexamples to diagnose them. For instance, *model checking* provides both an exhaustive search of all possible behaviours of the system, and exact, rather than approximate, quantitative results. As an example, Fehnker et al. [10] used the Uppaal model checker [1] to analyse basic qualitative properties of the AODV routing protocol. The authors of [10] were able to analyse systematically all network topologies up to five nodes. However, crucial aspects such as *passage of time* and *probabilities* were not considered in their analysis.

Statistical Model Checking (SMC)[20,21] is a trade off between testing and formal verification: it consists in performing an appropriate number of runs of the model under examination to check whether a given property is satisfied with a certain probability. Unlike an exhaustive approach, a simulation-based solution does not guarantee a correct result with a 100% confidence. It is only possible to bound the probability of making an error. More precisely, according to theoretical Chernoff-Hoeffding bounds, it is possible to estimate the number of runs that the simulator must perform: the higher is the precision required in the analysis and the greater must be the number of runs.

In the current paper we apply SMC-Uppaal [8] (release 4.1.19, July 2014), a statistical extension of the Uppaal model checker which supports the composition of timed and/or probabilistic automata. In SMC-Uppaal the user must fix two main statistical parameters, α and ε, both in the real interval $]0,1[$. The answer provided by the tool is a confidence interval $[p-\varepsilon, p+\varepsilon]$ for estimating the probability p of the desired property; α represents the probability of false negatives while ϵ is the probabilistic uncertainty. In the last two releases of SMC-Uppaal, the number of runs to be executed in a simulation to ensure a fixed precision is not estimated a priori anymore; instead it is continually re-computed during the simulation, taking into consideration the results of the runs executed up to that point. As a consequence, starting from SMC-Uppaal 4.1.18 there is a dramatic reduction of the average number of runs effectively executed in a simulation.

Our work has been strongly inspired by a recent comparison between the two ad hoc routing protocols AODV and DYMO, on arbitrary networks up to 5 nodes with perfect communication [12], relying on the SMC-Uppaal model checker (release 4.1.11). DYMO [18] is a recent evolution of AODV (since March 2012 it is sometimes referred to as AODVv2) that tries to populate the routing tables of each node by adopting a concept called *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. In principle this should result in better performances of the routing process. However, the analysis of [12] revealed that DYMO establishes fewer routes on average than does AODV. This calculation is obtained by counting the average number of entries appearing in the routing tables of all nodes after completing routing requests. Also the average quality of the routes found by AODV seems to be better than that of DYMO. Here route quality measure the difference between the length of the routes found by the routing protocol and the length of the corresponding optimal route.

These results cast a shadow on the more recent and more sophisticated DYMO protocol. Actually, it would seem that path accumulation in DYMO constitutes more a problem rather than an help. We conjecture that the results of [12] applies only to small networks, where the proliferation of extra messages may really constitute a problem. For this reason we decided to use the most recent release of SMC-Uppaal to repeat the analysis of [12] on networks of bigger size, operating in a slightly more realistic communication scenario. Ad hoc routing protocols have been developed for networks operating in harsh operating conditions. In particular, *communication failures* are quite common in MANETs: wireless communications can easily fail due to either communication collisions or environmental conditions such as temporary obstacles or physical interferences.

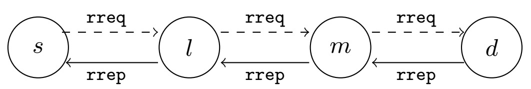
We have adapted the SMC-Uppaal models of [12] to compare AODV and DYMO on 4x3 *toroid topologies*, i.e. 4x3 grids circularly connected in the two dimensions. In this manner, each of the 12 nodes is connected with exactly 4 neighbours. We have adopted a *probabilistic model* of wireless communication to take into account *message loss* at different rates. For high loss rates this allows us to emulate *scarse networks*, i.e. networks scarsely connected.

As in [12] we consider three different workbenches to compare the two protocols: i) a probabilistic analysis to estimate the ability to successfully complete the protocol; ii) a quantitative analysis to determine the average number of routes found during the routing process; iii) a qualitative analysis to verify how good (i.e. short) are the routes found by the routing protocol. In our probabilistic analysis, in the case of perfect communication, AODV and DYMO have pretty much the same performances. However, with the introduction of some loss rate, DYMO performs dramatically better than AODV: up to 20% better than AODV, with a 30% loss rate. In the quantitative analysis DYMO performs at least 24% better than AODV. Again, the gap between the two protocols is wider when increasing the loss rate. Finally, our qualitative analysis shows that, in this respect, the two protocols behave pretty much in the same manner.

Outline In Section 2 we describe the two protocols under examination: AODV and DYMO. In Section 3 we recall and extends the SMC-Uppaal models of [12] for the two protocols. In Section 4 we repeat the experiments of [12] in our setting. The paper ends with a discussion of the results.

Figure 1 The AODV routing protocol.

$$\begin{array}{rcll}
s & \longrightarrow & * & : \quad \texttt{rreq}, s, Rid, d, Sseq, Dseq, 0 \\
l & \longrightarrow & * & : \quad \texttt{rreq}, s, Rid, d, Sseq, Dseq, 1 \\
m & \longrightarrow & * & : \quad \texttt{rreq}, s, Rid, d, Sseq, Dseq, 2 \\
d & \longrightarrow & m & : \quad \texttt{rrep}, s, d, Dseq', 0 \\
m & \longrightarrow & l & : \quad \texttt{rrep}, s, d, Dseq', 1 \\
l & \longrightarrow & s & : \quad \texttt{rrep}, s, d, Dseq', 2
\end{array}$$



2 AODV and DYMO: two different generations of ad-hoc routing protocols

This section provides a brief overview of both ad-hoc routing protocols.

AODV [17] is one the four protocols standardised by the IETF MANET working group. The protocol is intended to first establish a route between a source node and a destination node (*route discovery*), and then maintain a route between the two nodes during topology changes caused by node movement (*route maintenance*). Since AODV works on-demand, routers only maintain distance information for nodes reached during route discovery. In this paper we focus on the route discovery process.

In the AODV protocol each node maintains a *routing table* (RT) containing informations about the routes to be followed when sending messages to the other nodes of the network. In particular, for each destination node n a routing table provides an entry containing the following information: (i) the name of the *destination node* (say n); (ii) the number of *hops* necessary to reach n; (iii) the *neighbour node* in the route towards n; (iv) a *destination sequence number* to represent how fresh the information is: the higher the sequence number is, the fresher the path will be; (v) a validity *flag* for that entry. The collective information in the nodes' routing table is at the best a partial representation of network connectivity as it was sometimes in the past; in the most general scenario, mobility together with node and communication failures continually modify that representation.

Each node maintains also a *local history table* (HT) containing pairs of the form (*source-name*, *request-id*) to discard request packets which have already been processed.

In Figure 1, we report a scheme of the AODV protocol on a network of four nodes in a line topology: a source s, a destination d and two intermediate nodes l and m. We also provide a graphical representation of the flow of messages: dashed arrows denote the broadcast of *route request packets* (`rreq`), while continuous arrows denote the unicast sending of *route reply packets* (`rrep`). More precisely, suppose the source node s wishes to send a message to the destination node d. In order to perform the sending, s will look up an entry for d in its routing table. If there is no such an entry it will launch a route discovery procedure to find a route to d. The protocol works as follows:

- The source s broadcasts a route request packet of the form

$$\langle \texttt{rreq}, s, \mathit{Rid}, d, \mathit{Sseq}, \mathit{Dseq}, \mathit{hc} \rangle \ .$$

  Here, the fields s and d denote the IP addresses of source and destination, respectively. The field Rid denotes a *request-id*, that is a sequence number uniquely identifying the request. The Sseq field contains the *source sequence number*, i.e. the current sequence number to be used in routing table entries pointing towards the source node s. The Dseq field is the *destination sequence number* containing the latest sequence number received in the past by the source node s for any route towards the destination d; this number is 0 if d is unknown to s. The *hop-count* field hc keeps track of the number of hops from the source node to the node handling the request. Initially, this field is set to 0.
- When the intermediate node l receives the route request, it acts as follows:
  - It looks up the pair (s, Rid) in its local history table to verify whether the request has already been processed. If this is the case, the request is discarded and the processing stops. Otherwise, the pair is entered into the local history table, so that future requests from s with the same Rid will be discarded.
  - Then, l looks up an entry for d in its routing table. If there is such an entry, with destination sequence number greater than or equal to the Dseq, then a route reply packet is sent back to the source saying to use l itself to get to the destination d. Otherwise, it re-broadcasts the route request packet with the hc field incremented by one.
  - In any case, l compares the source sequence number Sseq contained in the request with the one appearing in its routing table associated with node s. If Sseq is more recent (i.e. greater) than the one in the table, l updates its routing table entry associated with s.
- Node m will repeat the same steps executed by node l.
- Whenever the destination d receives the route request, it sends to m a unicast reply packet of the form

$$\langle \texttt{rrep}, s, d, \mathit{Dseq}', \mathit{hc}, \mathit{lt} \rangle \ .$$

  Here, the source address and the destination address are copied from the incoming request, while the destination sequence number is possibly updated according to d's routing table. The *hop-count* field is set to 0. The *lifetime* field lt contains the time expressed in milliseconds for which nodes receiving the `rrep` consider the route to be valid.

- The reply packet then follows the reverse path towards node s increasing the hc field at each hop. Each node receiving the reply packet will update the routing table entry associated with d if one of the following conditions is met:
  - No route to d is known;
  - The sequence number for d in the route reply packet is greater than that stored in the routing table;
  - The sequence numbers are equal but the new route is shorter.

  In this way, nodes on the reverse route learn the route to d.

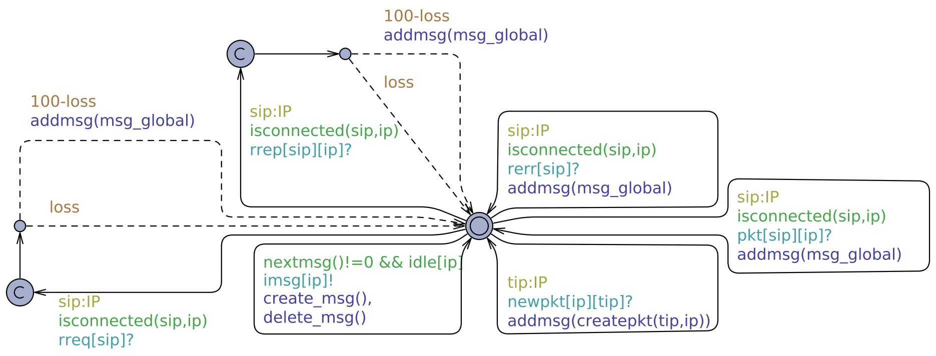
The architecture of the DYMO protocol [18] is quite similar to that of AODV. Here we follow the explanation of [12] to highlight only the major design differences between the two protocols.

- DYMO's mechanism for managing duplicate `rreq` messages is no longer based on checking the history table. Instead DYMO check the sequence number inside a route request to judge whether that request should be forwarded or discarded. While this modification save some memory, it has been shown that the change can lead to loss of route requests [9].
- On the other hand AODV can loose route replies since `rrep` messages are only forwarded if the routing table of an intermediate node is updated (changed). To avoid this, in DYMO a node generating a route reply increments the sequence number for the destination, thereby guaranteeing that the routing tables of nodes receiving the `rrep` message will be updated, and the `rrep` forwarded.
- DYMO establishes *bidirectional* routes between originator and destination. When an intermediate node initiates a route reply, it unicasts a message back to the originator of the request (as AODV does), but at the same time it forwards a route reply to the intended destination of the route request. In this manner the destination node gets all informations about intermediate nodes.
- DYMO uses the concept of *path accumulation*: whenever a control message travels via more than one node, information about *all* intermediate nodes is stored in the message. In this way, a node receiving a message establishes routes to *all other intermediate nodes*. In AODV nodes only establish routes to the initiator and to the sender of a message.

3 A probabilistic model for AODV and DYMO in SMC-Uppaal

In this section we provide a slight extension of the SMC-Uppaal models of [12] for both AODV and DYMO, where probabilities are introduced to model message loss. Both protocols are represented as parallel composition 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

Figure 2 `Queue`(`ip`) model for DYMO.

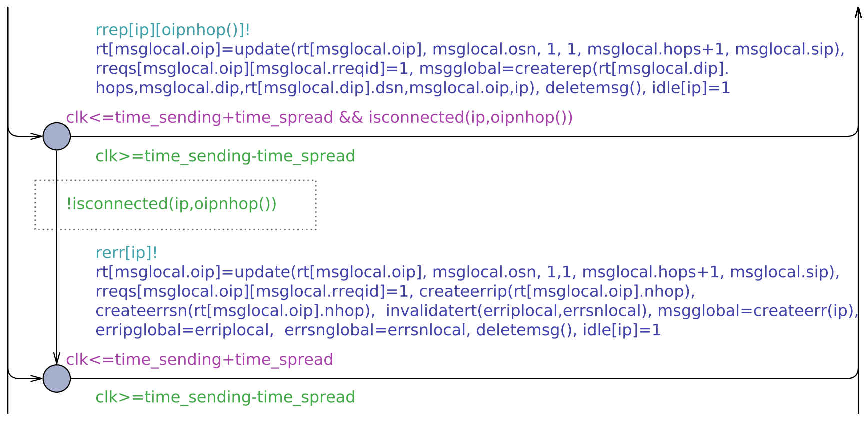


the handler depends on the type of the message. Communication between nodes `i` and `j` is only feasible if they are neighbours, i.e. in the transmission range of each other. This is modelled by predicates of the form `isconnected`[`i`][`j`] which are true if and only if `i` and `j` can communicate. 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`).

The `Queue` of a node `ip` for DYMO is depicted in Figure 2; the `Queue` automaton for AODV is very similar. 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 [12] our `Queue` is essentially a probabilistic timed automata. SMC-Uppaal 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 $(100-\text{loss})$. Notice that SMC-Uppaal requires input determinism to ensure that the system to be tested always produces the same outputs on any given sequence of inputs. Thus we need an extra intermediate committed location instead of branching immediately on the receiving action.

The `Handler` automaton, modelling the message-handling protocol, is far more complicated and has around 20 states. The implementation of the two protocols basically 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 AODV [17], the most time consuming activity is the communication between nodes, which take on average 40 milliseconds. 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 mil-

Figure 3 Extract from `Handle`(`ip`, `art`[`ip`]) model for AODV at `RREQ` message.


rrep[ip][oipnhop()]!
rt[msglocal.oip]=update(rt[msglocal.oip], msglocal.osn, 1, 1, msglocal.hops+1, msglocal.sip),
rreqs[msglocal.oip][msglocal.rreqid]=1, msgglobal=createrep(rt[msglocal.dip].
hops,msglocal.dip,rt[msglocal.dip].dsn,msglocal.oip,ip), deletemsg(), idle[ip]=1
clk<=time_sending+time_spread && isconnected(ip,oipnhop())
clk>=time_sending-time_spread
!isconnected(ip,oipnhop())
rerr[ip]!
rt[msglocal.oip]=update(rt[msglocal.oip], msglocal.osn, 1,1, msglocal.hops+1, msglocal.sip),
rreqs[msglocal.oip][msglocal.rreqid]=1, createerrip(rt[msglocal.oip].nhop),
createerrsn(rt[msglocal.oip].nhop), invalidatert(erriplocal,errsnlocal), msgglobal=createerr(ip),
erripglobal=erriplocal, errsnglobal=errsnlocal, deletemsg(), idle[ip]=1
clk<=time_sending+time_spread
clk>=time_sending-time_spread

liseconds is selected uniformly at random. Due to lack of space, we cannot present the full timed automaton modelling the `Handler`, but it is available online[3].

The `Handler` automata of the two protocols are exactly the same as those made available in [12] except for a few minor details. In particular, in the model for AODV of [12] we noticed a missing guard

$$!\texttt{isconnected}(\texttt{ip}, \texttt{oipnhop}())$$

on an arc that resulted in an `rerr`-message even if the networks was connected. As a consequence a node could get into a non-deterministic choice between sending a `rrep`-message and an `rerr`-message. In our model we have introduced the missing guard as depicted in Figure 3. Moreover, in order to correctly communicate the number of hops to the target node, we corrected the `rrep`-message `rrep[ip][msg_local.tip]!` as `rrep[ip][rt[msg_local.tip].nhop]!` in the `Handler` of DYMO. In both automata the constant `MAX_HOP_LIMIT` is now set to 100 instead of 10: in a 5-nodes network the value 10 is widely enough, but when working with bigger networks (we have worked also on 7x7 toroidals) this limitation may become a problem.

4 Experiments

We replay the experiments of [12] to compare AODV and DYMO on 4x3 toroidals (12 nodes) with possibly lossy channels. As in [12] we consider three different workbenches to compare the two protocols: i) a probabilistic analysis to estimate the ability to successfully complete the protocol finding the requested routes for a number of properly chosen scenarios; ii) a quantitative analysis to determine the average number of routes found during the routing process in the same scenarios;

[3] `http://www.profs.scienze.univr.it/~merro/nfm2015/`

iii) a qualitative analysis to verify how good (i.e. short) are the routes found by the routing protocol. Our experiment relies on 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) SMC-Uppaal 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 -yielding a confidence level of 99%. With these parameters SMC-Uppaal checks for each experiment a number of runs that can go from a few hundreds to 26492, in the worst case.

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 [12] (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 4x3 toroidal, we have 1728 different configurations. From this number we deduct 276 configurations because they make little sense in our analysis, as the source and the destination node coincide. This calculation yields 1452 different experiments. As we will repeat our simulations for three different loss rates, this makes in total 4356 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 millisecond, two of the distinct nodes receive a data packet and have to find routes to the packet's destinations. The query in SMC-Uppaal's 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. 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 within 10000 time units (milliseconds); as in [12] 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 on the AODV model. More precisely, we report the average probability to satisfy the required property in all 1452 different configurations. This is done for three different loss rates: 0%

Table 1 AODV: Probability analysis on 4x3 toroidals ($\alpha = \epsilon = 0.01$).

loss rate	avg probability	standard deviation	avg runs	standard deviation
0%	0.984	0.0036	583	1795
10%	0.746	0.130	11521	4486
30%	0.354	0.190	12875	2980

Table 2 DYMO: Probability analysis on 4x3 toroidals ($\alpha = \epsilon = 0.01$).

loss rate	avg probability	standard deviation	avg runs	standard deviation
0%	0.990	0.001	294	154
10%	0.818	0.090	9416	3851
30%	0.429	0.164	14571	2085

(perfect communication), 10% and 30%. 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 0.98. In the same analysis, paper [12] estimates at 0.99 the success rate for AODV on 5-nodes networks with arbitrary topologies. It should not surprise to see that the performances of AODV are strongly influenced by the message-loss rate. From a model-checking point of view, it is interesting to notice that the higher is the loss rate the greater is the number of runs required to complete the simulation. This is because with unreliable channels control messages need to be resent, making longer the whole routing process.

Table 2 presents the results for the same experiments on the DYMO protocol. In the case of perfect communication, our analysis shows that the probability of success in establishing the route requests can be estimated at around 0.99. In a similar analysis, paper [12] estimates success probably in DYMO, on arbitrary 4-nodes networks, at around 0.94.

Putting together the results of Tables 1 and 2 we can see that on 4x3 toroidals with perfect communication the reliability of the two protocols is quite similar. However, in the presence of message loss, DYMO performs much better than AODV. Actually, the higher is the loss rate the bigger is the gap between the two protocols. More precisely, with a 10% loss rate DYMO performs 10% better than AODV, whereas with a 30% loss rate DYMO performs 20% better then AODV. 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 [12] compares the performances of AODV and DYMO by taking 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. AODV does so only for the originator/destination node and for the sender of each message; whereas DYMO uses *path accumulation*

Table 3 Route quantity on 4x3 toroidals (26492 runs for each experiment).

	loss 0%	stand. dev.	loss 10%	stand. dev.	loss 30%	stand. dev.
AODV	62.30	2.79	60.89	3.11	54.79	3.48
DYMO	77.61	4.59	75.77	4.51	69.87	4.36
max	132	-	132	-	132	-

to establish routes to all intermediate nodes of a path. This difference in design between the two protocols should make a significative difference in the number of routes computed by the two protocols. However, the analysis made in [12], for all possible topologies up to 5-nodes, provides a quite surprising result: AODV establishes more routes on average than does DYMO. The authors have obtained their results by checking 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 the largest of these numbers among all runs of the simulation. 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 *value estimation*, as the result of the query is a value and not a probability. Since value estimation does not fix the statistical parameters α and ϵ, from which it is determined the number of runs, we set 26492 runs for our simulations to guarantee a 99% confidence.

We repeat the same analysis of [12] on our 4x3 toroidals by considering three different loss rates. In total we did 4356 experiments, one for each configuration with a different loss rate. The results of our analysis are reported in Tables 3. Note that the last row shows the maximal number of routing entries which can be involved during the routing processes: this number is $n \cdot (n-1)$ because in an n-node network each node has a routing table with $n-1$ entries. Tables 3 shows that during the routing process DYMO establishes on average 24.5% more routes than AODV, in the absence of message loss. This gap rises up to 27.5% with a 30% loss rate. It is quite interesting to notice that in both protocols the introduction of a loss rate has a relatively small influence on the average number of established routes.

In the same analysis of [12], on arbitrary networks up to 5 nodes without message loss, the results obtained depict a complete different picture: AODV establishes on average 15% more routes than DYMO.

4.3 Optimal routes

The results of the previous section tell us that in our 4x3 toroidals DYMO is more efficient than AODV in populating routing tables while establishing routing requests. In this section we provide a class of experiments to compare the

Table 4 Höfner and McIver's route quality on 4x3 toroidals (738 runs).

	loss 0%	stand. dev.	loss 10%	stand. dev.	loss 30%	stand. dev.
AODV	0.02%	0.14	1.65%	0.68	9.10%	2.50
DYMO	1.91%	1.24	6.03%	1.45	14.58%	1.69

ability of the two protocols in establishing optimal routes, i.e. routes of minimal length, according to the network topology. As explained in [12,16], all adhoc 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 obviously more evident in a scenario with an unreliable communication medium.

We start our analysis by replaying the same experiments of [12]. In particular, we check the property

```
E[<=10000,26492](max:quality())
```

for all possible configurations and loss rates. Again, this makes in total 4356 experiments. Here, the function `quality()` compares the length of the established routes with the length of the corresponding optimal routes. This is done by considering *all non-empty hops-entries of all routing tables of all nodes*. More precisely, for a given configuration, the property above returns the maximum among 738 runs (to ensure a 95% confidence) of the *average deviation* from the optimal route of all hops-entries. Then, as in [12], our experiment returns the average on all possible configurations. And as in [12], the results of Table 4 say that the established routes in AODV are significantly closer to optimal routes, when compared to DYMO. The gap between the two protocols goes from a couple of percentage points, in the case of perfect communication, up to 5 points, with a 30% loss rate. These results are not that surprising as the `quality`() function takes into consideration *all and only* non-empty entries, i.e. those entries which have been involved somehow in the routing process. As described in the previous section, DYMO, unlike AODV, fills routing entries of nodes which are not directly involved in the routing request. However, there is no guarantee that these entries are filled with optimal routes. Thus, if after two route requests AODV fills 62 entries while DYMO fills 78 entries, then the function `quality`() returns for AODV the maximum average deviation on 62 entries, while in DYMO it returns the maximum average deviation on 78 entries. We believe that the two protocols should be compared considering the same routing entries. In fact, the extra 16 non-empty entries in DYMO are not necessarily optimal but they are definitely closer to the optimal route when compared to the corresponding empty entries of AODV. Thus, perhaps the `quality`() function proposed by Höfner and McIver is not the best instrument to test which of the two protocols establish the better route as a result of a route request.

As a consequence, we decided to reformulate our analysis on route quality by making a different experiment. We checked the following property:

Table 5 Optimal routing on 4x3 toroidals ($\alpha = \epsilon = 0.01$).

	loss 0%	stand. dev.	loss 10%	stand. dev.	loss 30%	stand. dev.
AODV	0.980	0.042	0.696	0.119	0.280	0.161
DYMO	0.983	0.022	0.712	0.087	0.298	0.129

```
        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. As usual this property is checked on all 4356 configurations with three 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 $\alpha = \epsilon = 0.01$, as usual.

Table 5 says that the probability to establish optimal routes in the two routing protocols is very close. Actually, in the presence of message loss, there is a small gap, between 0.01 and 0.02, in favour of DYMO. This gap would become bigger if we would focus only on the optimality of the second route request, which is lauched slightly after the first one. This is because DYMO works better then AODV when routing tables are non completely empty.

5 Conclusions, Related and Future Work

The formal analysis of MANETs and their protocols is challeging and go beyond the usual requirements for standard network protocols. In particular, the formal verification of ad hoc routing protocols received a lot of attention from the formal methods community [3,5,14,10,11,12,2,15].

Our work has been strongly inspired by a recent comparison of AODV and DYMO on arbitrary 5-node networks, in the ideal case of perfect communication [12]. In that analysis the DYMO protocol does not seem to perform better than the ten years older AODV protocol. In our opinion, some of the negative results of [12] about the performances of DYMO are due to the fact that 5-node networks might be too small (or scarsely connected) to allow DYMO to beneficiate of *path accumulation*.

In our paper we have carried on the analysis of [12] on 4x3 toroidals with possible lossy communication. We have extended the models of [12] to our setting and obtained a network of probabilistic timed automata [7] which has been used for doing Statistical Model Checking within the UPPAAL toolset [8] (release 4.1.19, July 2014). As a main result, in contrast with the results of [12], we have showed that on 4x3 toroidals the performances of DYMO appear to be significantly better than those of AODV. In particular, the probability to satisfy a route request in DYMO is significantly higher than in AODV in the presence of message loss: DYMO performs up to 20% better than AODV, with a 30% loss rate. In the quantitative analysis DYMO performs at least 24% better than AODV, with both loss rates. Again, the gap between the two protocols becomes larger when increasing the loss rate. Finally, the quality analysis of the established routes is a bit more delicate. We believe that the function `quality`() designed in [12] is not appropriate to estimate the quality of the requested routes, because it gives the average deviation of all non-empty entries. So, we have proposed a different query which estimates the deviation from the optimal route of the paths obtained from the required route requests. The results say that both protocols are pretty good in finding optimal routes with very small differences, depending on the loss rate. Notice that our quality analysis starts always from scratch, with empty routing tables. We conjecture that in a scenario where routing tables are non-empty DYMO will do better than AODV, also in term of route quality.

As in [12] we have assumed stationary networks. It would be interesting to compare the two protocols in a scenario with node mobility, along the lines of the work done in [11]. Moreover, we would like to extend our analysis to sparse grids affected by an increasing number of node and/or link failures. It would be interesting to check whether the robustness of DYMO makes a difference in such a kind of networks.

In order to study bigger systems with an higher confidence, paper [4] proposes a distributed implementation of UPPAAL SMC. We are planning to employ this approach to extend our results to bigger networks.

Acknowledgements We thank Giacomo Annaloro and Marco Campion for their preliminary experiments on 7x7 toroidals. Annaloro and Campion also found some minor errors in the SMC-Uppaal models of [12], as pointed out in Section 3. We thank the anonymous referees for their constructive comments.

References

1. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: International Conference on the Quantitative Evaluation of Systems (QEST). pp. 125–126. IEEE Computer Society (2006)
2. Benetti, D., Merro, M., Vigano, L.: Model checking ad hoc network routing protocols: Aran vs. endairA. In: 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM). pp. 191–202. IEEE Computer Society (2010)
3. Bhargavan, K., , Obradovic, D., Gunter, C.: Formal verification of standards for distance vector routing protocols. Journal of the ACM 49, 538–576 (2002)
4. Bulychev, P.E., David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Checking and distributing statistical model checking. In: 4th NASA Formal Methods Symposium (NFM). Lecture Notes in Computer Science, vol. 7226. Springer (2012)

5. Chiyangwa, S., Kwiatkowska, M.: A timing analysis of AODV. In: Formal Methods for Open Object-Based Distributed Systems (FMOODS). Lecture Notes in Computer Science, vol. 3535, pp. 306–321. Springer Berlin Heidelberg (2005)
6. Clausen, T., Jacquet, P.: Optimized Link State Routing Protocol (OLSR) (2003), rFC 3626
7. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: Formal Modeling and Analysis of Timed Systems (FORMATS). Lecture Notes in Computer Science, vol. 6919, pp. 80–96. Springer (2011)
8. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 6806, pp. 349–355. Springer (2011)
9. Edenhofer, S., Höfner, P.: Towards a rigorous analysis of AODVv2 (DYMO). In: IEEE International Conference on Network Protocols (ICNP). pp. 1–6. IEEE Computer Society (2012)
10. 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)
11. Höfner, P., Kamali, M.: Quantitative analysis of AODV and its variants on dynamic topologies using statistical model checking. In: International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS). Lecture Notes in Computer Science, vol. 8053, pp. 121–136. Springer (2013)
12. Höfner, P., McIver, P.: Statistical model checking of wireless mesh routing protocols. In: 5th NASA Formal Methods Symposium (NFM). Lecture Notes in Computer Science, vol. 7871, pp. 322–336. Springer (2013)
13. Johnson, D.B., Maltz, D.A.: Dynamic Source Routing in Ad Hoc Wireless Networks. Kluwer Acad. Pub. (1996)
14. Liu, S., Ölveczky, P.C., Meseguer, J.: A framework for mobile ad hoc networks in real-time maude. In: 10th Rewriting Logic and Its Applications, Lecture Notes in Computer Science, vol. 8663, pp. 162–177. Springer (2014)
15. Merro, M., Sibilio, E.: A calculus of trustworthy ad hoc networks. Formal Aspects of Computing 25(5), 801–832 (2013)
16. Miskovic, S., Knightly, E.: Routing primitives for wireless mesh networks: Design, analysis and experiments. In: IEEE International Conference on Computer Communications (INFOCOM). pp. 2793–2801. IEEE Computer Society (2010)
17. Perkins, C., Belding-Royer, E., Das, S.: Ad-hoc on-demand distance vector (AODV). RFC 3561 (Experimental) (2003), `http://www.ietf.org/rfc/rfc3561`
18. Perkins, C., Chakeres, I.: Dynamic MANET on-demand (AODVv2) routing. IETF Internet Draft (Work in Progress) (2012)
19. Perkins, C.E., Bhagwat, P.: Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In: Conference on Communications Architectures, Protocols and Applications (SIGCOMM). pp. 234–244 (1994)
20. Sen, K., Viswanathan, M., Agha, G.: VESTA: A statistical model-checker and analyzer for probabilistic systems. In: International Conference on the Quantitative Evaluation of Systems (QEST). pp. 251–252. IEEE Computer Society (2005)
21. Younes, H., S.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon University (2004)