Mojgan Kamali | Massimo Merro | Alice Dal Corso

AODVv2: performance vs. loop freedom

Turku Centre *for* Computer Science

TUCS Technical Report
No 1177, February 2017

AODVv2: performance vs. loop freedom

Mojgan Kamali
Åbo Akademi University, Faculty of Science and Engineering,
the Agora building, 3rd floor, room 341A, Vesilinnantie 5, 20500 Turku, Finland
mojgan.kamali@abo.fi
Massimo Merro
University of Verona, Department of Computer Science,
Strada Le Grazie 15 - 37134 Verona, Italy
massimo.merro@univr.it
Alice Dal Corso
University of Verona, Department of Computer Science,
alice.dal_corso@studenti.univr.it

TUCS Technical Report
No 1177, February 2017

Abstract

In this paper, we focus on two different evolutions of Ad-hoc On-demand Distance Vector (AODV) routing protocol, i.e., DYMO and AODVv2. We investigate the performance of these protocols as well as check the protocols for loop freedom. We sketch how DYMO can cause routing loops whereas AODVv2 can overcome this issue by paying the price of degraded performance. Our modelling and analysis are carried out by the Uppaal Statistical Model Checker (SMC).

Keywords: Mobile Ad-hoc Networks, Reactive routing Protocol, Statistical Model Checking, Performance, Loop Freedom

TUCS Laboratory
Distributed Systems Laboratory

1 Introduction

Wireless Networks are on the rise from the more obvious laptops and smart phones in use everywhere, to sensor networks generating large amount of data, and to envisioned electrical cars being charged wirelessly. Ad hoc networking among wireless communication has gained popularity and is applied in a wide range of application areas, such as public safety, emergency response networks, etc. Mobile Ad-hoc Networks (MANETs) are self-configuring networks that support broadband communication without relying on any wired infrastructure. Therefore, they provide faster and low-cost network deployment. In these networks, nodes can move freely through the network causing the links to appear or to fail.

Routing protocols of ad-hoc networks are main factors determining the 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 the network. These routing protocols are divided into two main categories: reactive and proactive routing protocols. Reactive protocols find alternative routes on demand whenever needed whereas proactive protocols find different routes in advance by exchanging control messages through the network. Two examples of reactive and proactive routing protocols are Ad-hoc On-demand Distance Vector (AODV) [17] and Optimised Link State Routing (OLSR) [5], respectively.

In this work, we have focused on two evolutions of the AODV routing protocol to investigate their performance and their ability to cause routing loops. AODV 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 (*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 it is not feasible to simulate the systems for all possible scenarios. As a consequence, unexpected behaviour and flaws appear many years after the development of protocols. Since AODV works on-demand, routers only maintain distance information for nodes reached during route discovery, meaning as soon as a packet is injected into the system, the protocol starts finding routes whereas in proactive protocols this is not the case.

Proactive protocols find different routes in advance by exchanging control messages through the network which bears the benefit that routes to different destinations are available. As a consequence if a packet is injected into the network, it can rapidly be delivered to the destination.

In addition, the protocol's specification are written in English prose (available as RFCs) in order to provide the opportunity for the reader to read and understand the protocol behaviour. However this makes RFCs easy to understand, it may lead to ambiguities and different interpretations of the same specification causing to different implementations.

Formal methods as mathematical languages and techniques, complement alternatives such as test-bed experiments and simulation approaches, and provide valuable tools for designing and verifying MANET routing protocols. They allow the sketching of protocols in a precise manner and provide counter examples to diagnose their flaws. As such, they have a great potential for improving the precision of design and development. As a case in point, a model checking [4] technique is a powerful approach used for validating key correctness properties in finite representations of a formal system model. Statistical Model checking (SMC) [20] is a technique combining testing techniques and formal verification. 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, a simulation-based technique does not assure a 100% correct result, but it is possible to restrict the probability of an error occurring. In other words, it is possible to specify the number of runs that the simulator must perform to guarantee the level of required precision; as the higher precision of the analysis is required, the number of runs increases.

In this work, we apply Uppaal SMC [7], the extension of the Uppaal model checker [1], which allows the automata to be probabilistic. In Uppaal SMC the two main statistical parameters α and ϵ in the interval [0, 1] must be specified. 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. Our work has been strongly motivated by the recent version of the AODVv2 Internet draft [19] in which there are several modifications to the protocol to overcome the looping problem of AODV and DYMO (DYMO is the evolution of AODV protocol that has better performance due to the path accumulation [6]). Loop freedom is a critical and challenging property for any routing protocol, especially routing protocols of MANETs. A loop in the routing table is an established route stored in the routing table at a specific point in time that visits the same node more than one time before the intended destination is reached [9]. Caught packets in a routing loop can saturate the links and decrease the network performance. Several studies have shown that AODV and DYMO suffer from routing loops [3, 8, 12, 16].

In previous versions of AODV specification, when a node sends a request to find a path to a specific destination, the receiving intermediate nodes could send the reply back to the route request originator (if they have valid information about the destination node) and provide the information about the path to the destination node (route reply messages from intermediate nodes were the main factors causing routing loops). In the current version of AODV specification, no intermediate node is able to send the route reply back to any route request even if valid information about the destination of the route request is provided in the intermediate node, meaning that the route request has to travel to the destination node and the destination is only responsible for sending back the route reply. This behaviour will increase the time needed for route discovery (routing tables in AODVv2 are

not updated as often as they are updated in DYMO), leading to decrease the performance of the protocol.

Contributions: Our first contribution is to model the core functionality of the AODVv2 protocol [19]. Second, we investigate the performance of DYMO[1] and AODVv2 in terms of route discovery, number of route entries, optimal route finding, and packet delivery. As the third contribution, we analyse the ability of both protocols to create routing loops. We show how the modifications of AODVv2 help the protocol to have no loop by paying the price of degraded performance.

Outline: The paper is structured as follows: in Section 2, we overview the two generations of the AODV protocol and in Section 3 we briefly discuss the Uppaal models of the two protocols based on their English specifications ([18] and [19]). Sections 4 and 5 are the core of our paper where we present the results of our analysis w.r.t. the performance and loop analysis, respectively. We draw conclusions and review related work, as well as propose future research directions, in Section 6.

2 DYMO and AODVv2: two generations of reactive routing protocols

This section provides a brief overview of DYMO and AODVv2 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. 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 recent the information is: the higher the sequence number is, the more recent the path will be; (v) a validity *flag* for that entry. 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.

In Figure 1, we report a scheme of the DYMO 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

[1] We use our modified DYMO model, originally developed in [13], for our experiments. The modified model is available in `http://users.abo.fi/mokamali/FORTE2017`.

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

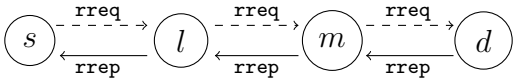

Figure 1: The DYMO routing protocol.

order to perform the sending, s will look up an entry for d in its routing table. If there is no such entry it will launch a route discovery procedure to find a route to d. The protocol works as follows:

- The source s generating a route request packet increases its sequence number and broadcasts the route request packet of the form $\langle \texttt{rreq}, s, d, Sseq\ Dseq, intms, hc \rangle$. Here, the fields s and d denote the IP addresses of source and destination, respectively. 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 field $intms$ denotes *intermediate* nodes on the way (accumulated path) from the message originator to the receiving node. DYMO uses the concept of *path accumulation*: whenever a control message travels via more than one node, information about *all* intermediate nodes is stored in the message. In this way, a node receiving a message establishes routes to *all other intermediate nodes*. Initially, there is no entry when the source broadcasts `rreq`. The path is accumulated later when the `rreq` is rebroadcast via intermediate nodes. 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:

  1. It looks up the pair (s, $Sseq$) in its routing 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 routing table, the routing table is updated for s and corresponding $intms$, so that future requests from s with the same $Sseq$ will be discarded.

2. 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 l increases its sequence number and a route reply packet is sent back to the source and to the destination. By this, 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, 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 information about intermediate nodes. If l does not have any entry with a destination sequence number greater than or equal to the $Dseq$, it adds itself as the intermediate node of `rreq` and then re-broadcasts the route request with the hc field incremented by one.

- Node m will repeat the same steps executed by node l.

- Whenever the destination d receives the route request, it increases its sequence number first and then sends to m a unicast reply packet of the form $\langle \texttt{rrep}, s, d, Dseq', intms, hc \rangle$. 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 sequence number which is unique to d. The *intms* represents intermediate nodes from the message originator toward the receiving node. Initially, there is no entry and the path is accumulated later when the `rrep` is unicast via intermediate nodes. The *hop-count* field is set to 0.

- The 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 and $intms$ if one of the following conditions is met: (i) no route to d is known; (ii) the sequence number in the route reply packet is greater than that stored in the routing table; (iii) the sequence numbers are equal but the new route is shorter. In this way, nodes on the reverse route learn the route to d.

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 destination.

The architecture of the AODVv2 protocol [19] is quite similar to that of DYMO considering some differences. Here we highlight the design differences between the two protocols.

- AODVv2's mechanism for managing duplicate `rreq` messages is based on checking the local route message set. This set contains the information about recently received `rreq`, e.g., source address, sequence number, destination address and metric. Whenever AODVv2 receives a `rreq`, it compares the newly received messages with this set to decide whether that request should be processed or discarded.

- AODVv2 establishes *bidirectional* routes between originator and destination. One of the main difference of AODVv2 is to avoid sending `rrep` by intermediate nodes. When AODVv2 broadcasts a `rreq`, it waits to get the `rrep` back only from the destination of the `rreq`. It means that intermediate nodes do not send the `rreps` to the source of the `rreq` even if they have active routes through the destination node. This behaviour will increase the time needed for route discovery (routing tables in AODVv2 are not updated as often as they are updated in DYMO), decreasing the performance of the protocol.

2.1 Degrading performance to avoid routing loops

In this section, we sketch how the two protocols differ in terms of performance and loops by providing an example of a potential loop in DYMO. We also discuss how AODVv2 protocol pays the price to remain loop free. Different studies have proved the presence of loops in DYMO protocol [16]. Here, we draw a simple example to show how a loop can appear in DYMO.

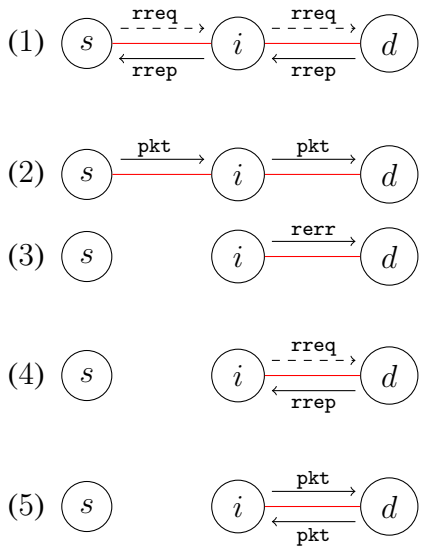

Figure 2: Presence of a loop in DYMO.

The network in Table 2 consists of three nodes that are connected in a linear topology (the red lines depict the connectivity between nodes). 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. 2(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. 2(1), updates its routing table for i and d and sends the pkt to node i to be delivered to d, Fig. 2(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. 2(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. 2(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. 2(5).

Protocol designers have overcome the looping problem of DYMO by incorporating several changes in the new version (AODVv2). 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, 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." [[19], page 28]

Here, LocalRoutes stores the previously received messages, AdvRte contains the information about newly received message, and LoopFree(AdvRte, LocalRoute):= (Cost(AdvRte) <= Cost(LocalRoute)).

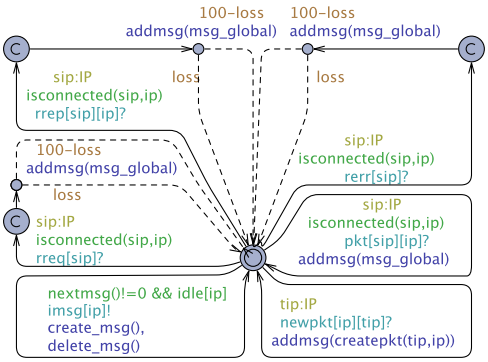

Figure 3: `Queue(ip)` model.

There are more conditions in the specification of the AODVv2 protocol indicating when to update routing tables, leading to less information being stored in the routing tables, hereby decreasing the performance. For instance, routing tables in AODVv2 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, whereas updating in this situation helps to repair the broken paths. In addition, the sending of `rrep` by intermediate nodes is not specified in AODVv2. 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 and DYMO

In this section, we briefly explain our AODVv2 automata and provide some modifications of the Uppaal SMC model of [13] for DYMO. 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`).

The `Queue` of a node `ip` for the three protocols is depicted in Figure 3. Messages from other nodes are stored in the queue. Only messages sent by nodes within the transmission range may be received. Unlike the model of [13] 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, i.e., `rreq`, `rrep` and `rerr`, with weight loss or receive them with weight $100-\mathsf{loss}$.

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

Based on DYMO and AODVv2 specifications, `rreq`s can be resent the maximum of 3 times in the presence of message loss. We have considered this behaviour when modelling AODVv2 in Uppaal. The major difference of AODVv2 that distinguishes it from DYMO, is the absence of using intermediate `rrep`s and also updating the routing tables. As we explained in Section 2, AODVv2 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). We have considered this behaviour when modelling AODVv2 in Uppaal[2].

Changes w.r.t. DYMO automaton:

The modified DYMO `handler` is depicted in Fig 4.

- In the DYMO model by [13], two connected nodes could get disconnected while a node is waiting to transmit a message (waiting time of 40 ms), which could cause a potential deadlock in the system. For our experiments, we modify this behaviour and assume that two connected nodes cannot get disconnected during this period of time which is the case in reality (the probability that two nodes disconnect upon communication is too low).

- We minimised the DYMO automaton of [13] by removing a number of redundant locations and transitions that were modelling the same procedure, Fig 4 (1,9, 11, 15).

[2]The reader can consult our models at `http://users.abo.fi/mokamali/FORTE2017`

- 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, Fig 4 (2).

- 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 Fig 4 (7, 14).

- We changed the guards of transition (3) in Fig 4 since there was a contradiction in the guards of the transition (`rt[msg_local.tip].nhop==0 && rt[msg_local.tip].nhop!=0`).

- We removed condition `rt[msg_local.tip].nhop==0` from the guards of transitions (4, 5) in Fig 4 as the condition was redundant. In other words, if `rt[msg_local.tip].nhop==0`, meaning that the next node along the path to the destination has not been updated previously which is equivalent to `!rt[msg_local.tip].f_flag` (valid route has not been found yet).

- We changed the type of `deliver` variable from boolean to an array with integer type to keep track of delivered packets used in our experiments Fig 4 (6).

- We modified the guards of transitions (8, 10, 13) Fig 4. This was due to the fact that the connectivity between nodes was not checked properly. When a destination or intermediate node is replying back to the source of the `rreq`, the connectivity between the that node and next node along the path to the originator of the `rreq` must be checked.

- We modified the synchronisation channels on the transitions (8, 12) since the node that is replying back has to be synchronised with the next node along the path to the originator of the `rreq`.

- We modified the update of variable `pending`. This variable is set to 1 for the `rreq` destination when a `rreq` is broadcast through the network and it is assigned to 0 later when the `rrep` was sent back to the originator of the `rreq` in transition (16) in Fig 4. In the previous model, variable `pending` was assigned to 1 for the `rreq` destination and was assigned to 0 later for the `rrep` destination which is the originator of the `rreq`, meaning that this variable was not updated for the corresponding node. In the current model, We assign variable `pending` to 1 for the `rreq` destination when broadcasting a `rreq` and later we assign 0 to this variable for the `rrep` originator which is the destination of the `rreq`.

- We changed the type of `msg_global`, `ips_global`, `sqns_global` and `dists_global` variables from meta variables to the types that we had defined for our model.

4 Performance analysis on static grids

We replay the experiments of [13] to compare DYMO, and AODVv2 on a 3x3 grid (9 nodes) with possibly lossy channels as well as one more property, namely the packet delivery property. We consider four 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; (iii) a qualitative analysis to verify how good (i.e. short) are the routes found by the routing protocol. (iv) 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 0.05 -yielding a confidence level of 95%. For each experiment with these parameters, Uppaal SMC checks several hundred runs of the model, up to 738 runs in the worst case. We run our experiments for the message loss rates used in [6], 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 [13] (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 experiments. This gives 10656 experiments in total for each protocol, since we repeat simulations for each combination of four loss rates and four experiments.

	loss=0%	stand. dev.	loss=10%	stand. dev.	loss=30%	stand .dev.	loss=40%	stand. dev.
DYMO	0.951	0.000	0.951	0.000	0.886	0.058	0.650	0.145
AODVv2	0.951	0.000	0.950	0.001	0.718	0.147	0.449	0.201

Table 1: Probability analysis on 3x3 grid-static network ($\alpha = \epsilon = 0.05$).

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. 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 (ms); as in [13] 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 two models. More precisely, we report the average probability to satisfy the required property in all 666 different configurations. This is done for the 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 0.95. We should add here that increasing the 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.

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. 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, whereas with 30% and 40% loss rate the gap between two protocols becomes more obvious (DYMO performs much better than AODVv2). 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.

	loss 0%	stand. dev.	loss 10%	stand. dev.	loss 30%	stand. dev.	loss=40%	stand. dev.
DYMO	37.268	7.678	37.416	6.189	34.661	5.891	31.247	5.461
AODVv2	34.564	5.945	34.393	5.744	34.548	5.928	31.616	5.436

Table 2: Route quantity on 3x3 grid-static network (738 runs for each experiment).

4.2 Number of route entries

The second analysis proposed in [13] 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 do that by checking the property

```
E[<=10000,738](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 738 runs for our simulations to guarantee a 95% confidence level.

We repeat the same analysis of [13] 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 (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 (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 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 [13, 15], 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 an unreliable communication medium.

We replay the same experiments of [13]. 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 $\alpha = \epsilon = 0.05$.

	loss 0%	stand. dev.	loss 10%	stand. dev.	loss 30%	stand. dev.	loss=40%	stand. dev.
DYMO	0.911	0.184	0.840	0.177	0.672	0.163	0.446	0.162
AODVv2	0.914	0.178	0.826	0.175	0.587	0.175	0.377	0.188

Table 3: Optimal routing on 3x3 grid-static network ($\alpha = \epsilon = 0.05$).

Table 3 says that the probability to establish optimal routes in the two routing protocols is very close when having no message loss.

Actually, in the presence of message loss, there is still a gap in favour of DYMO. This gap would become bigger if we would focus only on the optimality of the second route request, which is launched slightly after the first one. This is because DYMO works better than AODVv2 when routing tables are non 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 any 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:

	loss 0%	stand. dev.	loss 10%	stand. dev.	loss 30%	stand. dev.	loss=40%	stand. dev.
DYMO	0.951	0.000	0.951	0.000	0.784	0.099	0.503	0.162
AODVv2	0.893	0.126	0.858	0.119	0.423	0.097	0.195	0.078

Table 4: Packet delivery on 3x3 grid-static network ($\alpha = \epsilon = 0.05$).

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

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 the two packets are delivered at their destinations. Here, `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 within 10000 time units (ms); as in [13] 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 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 are not updated regularly due to the more restricted routing table updated in AODVv2. Therefore, the probability that all packets are delivered to the destination nodes is lower in AODVv2.

5 Loop analysis on grids with link breakage

We run our experiments w.r.t. the looping property for a 3x3 grid where links between nodes can break. We model link breakage by modifying the `Queue` automaton (depicted in Fig. 5) 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`. We assign this value to 80, so that with high probability the link between the sender and the receiver fails as link breakage is one of the main factors causing routing loops.

We consider 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;

	loss 0%	loss 10%	loss 20%	loss 30%	loss 40%
DYMO	1	2	2	2	2
AODVv2	0	0	0	0	0

Table 5: Number of loops in different configurations

(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 pkts to D.

The 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. To more carefully analyse loops in the network, we also run our experiments for a 20% message loss rate in addition to the four message loss rates we used for performance analysis. This gives 3330 experiments in total for each protocol. For simplicity we maintain the same number of runs as for performance analysis, i.e., 738. Our experiments can be represented using the following Uppaal SMC syntax:

```
E[<=10000;738](max:numberofloops())
```

Function `numberofloops()` returns an integer value which reflects the number of loops found in the network. Table 5 depicts the maximum number of loops considering different message loss rate in different configurations for both protocols.

The results show that when message loss rate increases, 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 as there is no routing loop while verifying AODVv2.

	loss 0%	loss 10%	loss 20%	loss 30%	loss 40%
DYMO	10	11	35	13	11
AODVv2	0	0	0	0	0

Table 6: Number of configurations that have loops

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 20%, the number of configurations that have loops

goes up to 35. The table depicts when message loss increases to 30% and 40%, the number of configurations that have loops respectively decrease to 13 and 11. In contrast to DYMO, there is no configuration in AODVv2 that has routing loops. The results show that AODVv2 does not cause any routing loops.

6 Conclusion, Related Work and Future Directions

Formal analysis of MANTEs and their protocols is a challenging task, however their formal verification have attracted the attentions from formal methods community [2, 6, 11, 13, 14, 16]. There are several studies on loop freedom of AODV and DYMO. Glabbeek et al. [12] 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. [16] 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 [3, 8, 10]. Our work complements the performance analysis of [6] where it was shown that DYMO has better performance than AODV due to the path accumulation of DYMO.

Our work has been strongly inspired by the recent version of AODVv2 Internet draft [19] where there are several modifications to the protocol to overcome the looping problem of DYMO. 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 [19] and compare the performance of DYMO and AODVv2.

In this paper we modelled the AODVv2 protocol and investigated the performance of two evolutions of the reactive protocols DYMO and AODVv2 in a 3x3 grid with possibly lossy communication as well as checking the loop freedom property for both protocols. Our analysis is performed using the Uppaal SMC. We have provided an Uppaal model which is in accordance with the AODVv2 standard, modelling core functionality of the protocol. We were able to show how the performance of the more recent AODVv2 has been worsened compared to the preceding DYMO to achieve a loop free routing protocol, i.e., AODVv2.

We investigated the performance of both protocols considering four different properties and four different message loss rates. DYMO performs better than the more recent AODVv2, however the results show that DYMO can cause routing loops whereas AODVv2 is loop free. The performance analysis is carried out for stationary networks where there is no link breakage through the network. However, we have verified the looping property of the protocols for the networks where links between nodes can fail.

Due to the application diversity of MANETs, different protocols have been developed. These protocols have been already implemented and deployed. However, it is unclear which protocol should be used in certain circumstances. To

answer these questions systematically, we aim at formally defining properties that can be used as measurements for routing protocols. To evaluate the measurements, we focus on comparing reactive and proactive protocols.

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 (2004)

[2] Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. J. ACM 49(4), 538–576 (2002)

[3] 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)

[4] Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)

[5] Clausen, T., Jacquet, P.: Optimized link state routing protocol (OLSR). RFC 3626 (Experimental) (2003)

[6] 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)

[7] David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Proceedings of the 23rd International Conference on Computer Aided Verification. pp. 349–355 (2011)

[8] 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)

[9] 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)

[10] 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)

[11] van Glabbeek, R., Höfner, P., Portmann, M., Tan, W.L.: Modelling and verifying the aodv routing protocol. Distributed Computing 29(4), 279–315 (2016)

[12] 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)

[13] Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: NASA Formal Methods Symposium (NFM'13). pp. 322–336 (2013)

[14] 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)

[15] Miskovic, S., Knightly, E.W.: Routing primitives for wireless mesh networks: Design, analysis and experiments. In: INFOCOM, 2010 Proceedings IEEE. pp. 1–9 (2010)

[16] 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)

[17] Perkins, C., Belding-Royer, E., Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental) (2003)

[18] Perkins, C., Stan, R., Dowdell, J.: Dynamic MANET On-demand (AODVv2) Routing draft-ietf-manet-dymo. Internet Draft 26 (2013)

[19] Perkins, C., Stan, R., Dowdell, J., Steenbrink, L., Mercieca, V.: Dynamic MANET On-demand (AODVv2) Routing draft-ietf-manet-aodvv2. Internet Draft 16 (2016)

[20] Sen, K., Viswanathan, M., Agha, G.A.: Vesta: A statistical model-checker and analyzer for probabilistic systems. In: QEST. pp. 251–252 (2005)

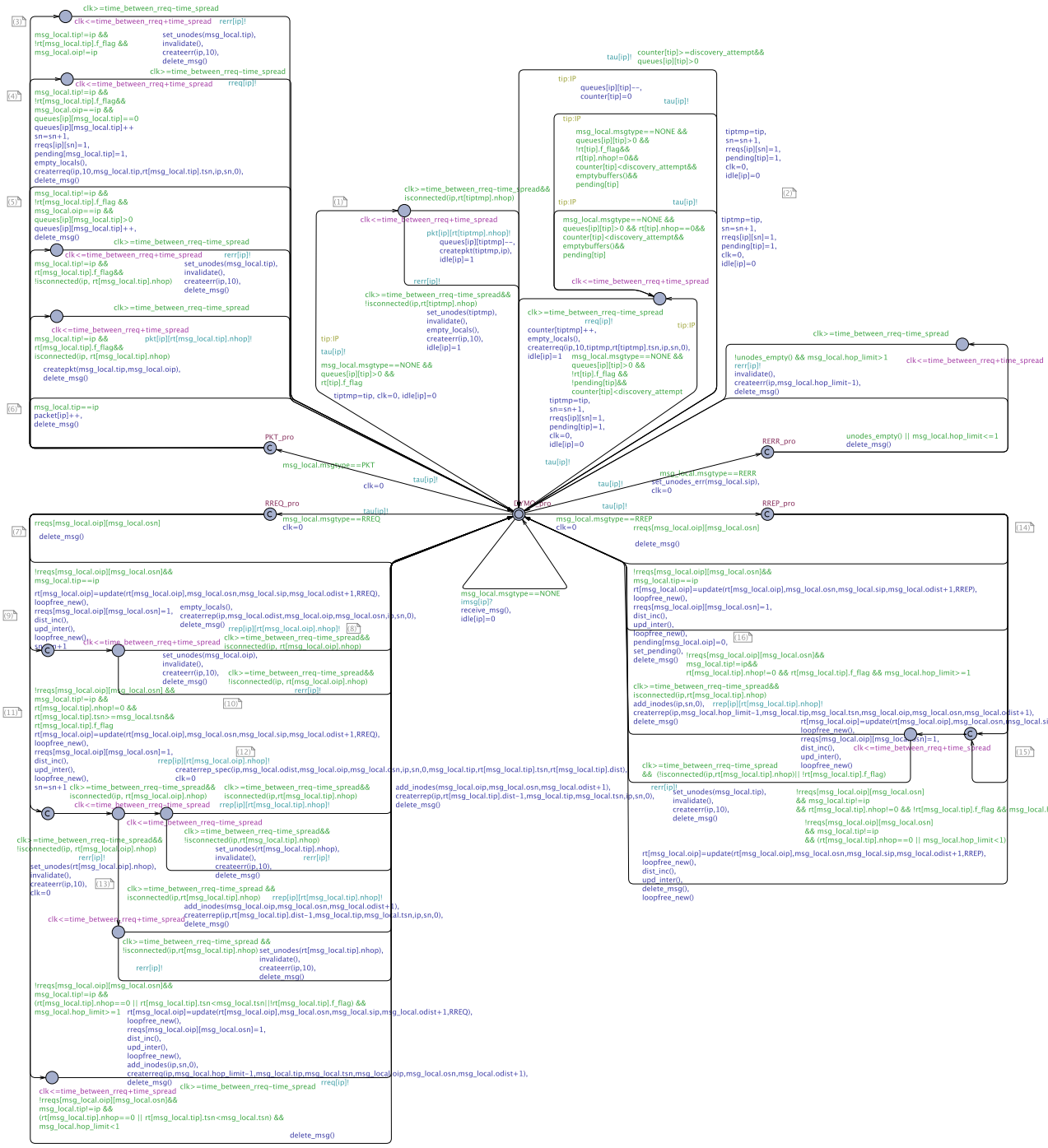

Figure 4: `handler` automaton.

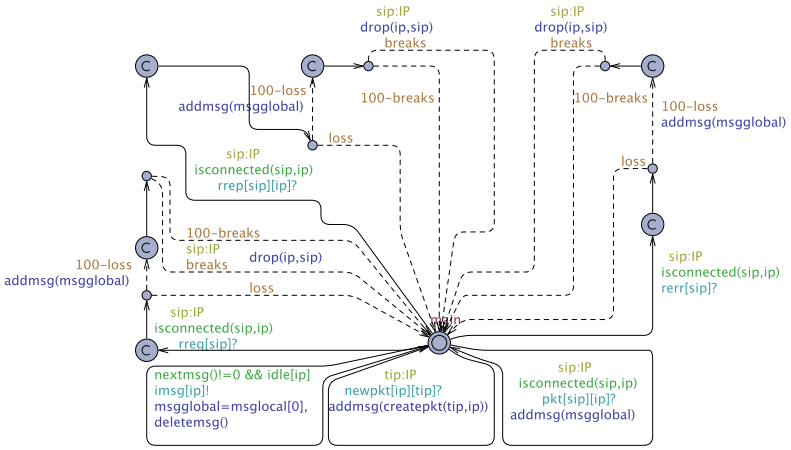

Figure 5: `Queue(ip)` model with link breakage.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 A, 20520 TURKU, Finland | www.tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

ISBN 978-952-12-3521-4
ISSN 1239-1891