A Timing Analysis of AODV

Sibusisiwe Chiyangwa and Marta Kwiatkowska

School of Computer Science
The University of Birmingham
Edgbaston, Birmingham, B15 2TT, United Kingdom
{syc,mzk}@cs.bham.ac.uk

Abstract. Mobile ad hoc networks (MANETs) are wireless networks formed spontaneously. Communication in such networks typically involves multi-hop relays, and is subjected to dynamic topology changes and frequent link failures. This complex scenario demands robust routing protocol standards that ensure correct and timely delivery of messages. Recently, formal verification has been successful in detecting ambiguities in protocol standards. We consider the Ad hoc On Demand Distance Vector (AODV) protocol, a reactive protocol currently undergoing standardisation at the IETF (RFC3561). AODV performs route discovery whenever a route to the destination is needed, and retains routing information for a period of time specified by the standard. We apply the real-time model checker UPPAAL to consider the effect of the protocol parameters on the timing behaviour of AODV, thus complementing the earlier untimed verification effort. Our study of the recent versions of the standard (RFC3561-bis-01) has highlighted a dependency of the lifetime of routes on network size, which can be alleviated by allowing the route timeouts to adapt to network growth.

1 Introduction

Mobile ad hoc networks (MANETs) are networks of mobile devices that communicate without the need for a central authority and infrastructure, formed without a-priori knowledge or planning. Ad hoc networks can be set up anywhere and anytime, are dynamic and often exhibit frequent topology changes due to loss of contact and movement. Three basic approaches to routing are taken: proactive protocols, which continuously exchange routing information between the nodes (e.g. OLSR); reactive, which build routes on demand (e.g. AODV); and hybrid combinations of the two (e.g. ZRP).

The Ad hoc On Demand Distance Vector (AODV) protocol [6] is a reactive routing protocol currently in the process of being standardized at the IETF (RFC3561) and implemented [12, 19]. To transmit data over such a network, the AODV protocol enables dynamic, multihop routing between devices. AODV is an *on demand* algorithm, meaning that a route discovery mechanism is invoked only when the sender wishes to transmit data. These routes are maintained as long as they are needed by the senders, and are deleted after a certain amount of time has passed so as not to overload the routing tables. AODV is designed

for ad hoc networks of a wide range of sizes, from the very small to networks of tens to thousands of mobile, Internet enabled, nodes. Simulation experiments for 1000 nodes have been reported, and first implementations are available [19].

The dynamic aspects of mobile ad hoc networks mean that both the topology of such networks and their size varies over time, giving rise to an unbounded execution tree and *infinitely many* states. This scenario is much more complex than for existing network protocols, and consequently designing protocols that achieve correct and timely delivery of messages is inherently more difficult. Much valuable effort is therefore being directed towards the formulation of routing protocol *standards* for MANETs, of which AODV RFC3561 is one example, to serve as a *specification* to which a protocol *implementation* must conform. The implementations are then developed according to the guidelines set by the standard. Unfortunately, the resulting protocol complexity sometimes results in unintentional ambiguities introduced into the standard, which, if undetected, can be transferred to the implementation. An analysis of the proposed standards is therefore desirable, and it results in subsequent revisions.

Recently, formal verification has been successfully employed as an aid to detect ambiguities in the proposed AODV standards and implementations [9, 15], resulting in the discovery of routing loop errors in early protocol versions (version 4) that have been addressed in later revisions of the draft standard. Both these approaches do not model real-time, and instead replace the real-valued timer events with non-deterministic time-outs. This can result in false positives, i.e. error traces that do not correspond to realistically timed scenarios, and is undesirable since the AODV protocol uses real-valued timers in an essential way, for example to determine the lifetime of routes. It is important that routing is handled in a timely manner, i.e. route discovery and message delivery happen without unnecessary time delays. The timing values are determined by formulas dependent on protocol parameters (constants) specified by the standard. Clearly, the choice of the constants and the route lifetimes will affect the timeliness of protocol actions, especially as the network size and topology change dynamically over time.

In this paper, we complement the existing analyses of the AODV protocol by model checking its timing aspects. Working from the most recent draft standard documents, we build a timed automata model for AODV using the UPPAAL [22, 24] model checker. We consider the effect of the default protocol parameters on the timing behaviour of AODV, and investigate properties such as timely route discovery and the ability to deliver messages within a specified time period. Our study of the AODV draft standard has highlighted a dependency of the lifetime of routes on network size, which may lead to failure to discover the route if it exists or failure to deliver the data to destination. The observation pertains to the latest version (RFC3561-bis-01 [6]) and, in a simpler form, to earlier versions (13 and RFC3561-bis-00) of the draft standard. Having inspected a recent implementation of AODV [19], we confirm our observation also for this implementation with the help of an ns-2 simulation experiment. We propose a

modification to the standard that alleviates the problem by allowing the route timeouts to adapt to network growth.

2 The AODV Protocol

The Ad Hoc On-Demand Distance Vector (AODV) Protocol is an IP routing protocol that allows users to find and maintain routes to other users in the network. AODV is *on-demand*, or *reactive*, in the sense that routes are established only when needed. The routing decisions are made using distance vectors, i.e. distances measured in hops to all available routers. The protocol supports unicast, broadcast, and multicast. The version of AODV we describe below is based on the RFC draft standard [6].

Each nodes maintains a *sequence number*, which saves a time stamp, and a *routing table*, which contains routes to destinations. Sequence numbers are used to determine the freshness of routes (the higher the number, the fresher the route, and the older one can be discarded). Each table entry contains the address of the next hop (next node to destination), a hop count (number of hops to the destination) and a destination sequence number. Since this is an on-demand distance vector scheme, routers maintain distances of those destinations only that they need to contact or relay information to. Each active route is associated with a lifetime stored in the table; after this time has passed route timeout is triggered, and the route is marked as invalid and later on removed. AODV uses two main procedures, *route discovery* and *route maintenance*, which are described below.

Route Discovery. If a sender (source node) needs a route to destination, it broadcasts a *ROUTE REQUEST* (RREQ) message. Every node also maintains a *broadcast_id* which, when taken together with the originator's IP address, uniquely identifies a RREQ. Every time a sender issues a RREQ, it increments its broadcast_id and sequence number by one. The sender buffers this RREQ for *PATH_DISCOVERY_TIME* (PDT) so that it does not reprocess it when its neighbours send it back. The sender then waits for *NET_TRAVERSAL_TIME* (NETT) for a *ROUTE REPLY* (RREP). If a RREP is not received within this time, the sender will rebroadcast another RREQ up to *RREQ_TRIES* times. With each additional attempt, the waiting time (NETT) is doubled.

When a node receives a RREQ message it has not seen before, it sets up a reverse route back to the node where the RREQ came from. This reverse route has a lifetime value of *ACTIVE_ROUTE_TIMEOUT* (ART). The reverse route entry is stored along with the information about the requested destination address. If the node that receives this message does not have a route to the destination, it rebroadcasts the RREQ. Each node keeps track of the number of hops the message has made, as well as which node has sent it the broadcast RREQ. If nodes receive a RREQ, which they have already processed, they discard the RREQ and do not forward it.

If a node has a route to the destination, it then replies by unicasting a RREP back to the node it received the request from. The reply is sent back to the sender

via the reverse route set by the RREQ. As the RREP propagates back to the source, nodes set up forward pointers to the destination. Once the source node receives the RREP, the route has been established and data packets may be forwarded to the destination.

Route Maintenance. The role of route maintenance is to provide feedback to the sender in case a router or link has gone down, to allow the route to be modified or re-discovered. A route can stop working simply because one of the mobile nodes has moved. If a source node moves, then it must rediscover a new route. If an intermediate node moves, it must inform all its neighbours that needed this hop. This message is forwarded to all the other hops and the old route is deleted. The source node must then re-discover a new route.

One proposed way for a node to keep track of its neighbours is by using *HELLO* messages. These are periodically sent to detect link failures. Upon receiving notification of a broken link, the source node can restart the rediscovery process. If there is a link breakage, a *ROUTE ERROR* (RERR) message can be broadcast on the net. Any host that receives the RERR invalidates the route and rebroadcasts the error messages with the unreachable destination information to all nodes in the network.

3 Correctness Requirements for Routing Protocols

Reactive routing protocols for mobile ad hoc networks are complex schemes for the following reasons. Firstly, the scheme must allow for an unbounded number of nodes acting in parallel, with each node acting as a router, destination and relay node. Secondly, the topology can change dynamically, and hence the protocol must be able to recover from link failures. Finally, real-time clocks play a key role in the protocol, both in setting the lifetimes of routes and triggering timeouts. Achieving efficient and correct routing in such scenarios is a non-trivial undertaking for standardisation efforts. The main correctness requirements for a routing protocol, first stated in [27], are:

I. If a path exists between two nodes, then a route between them will eventually be discovered.
II. When a route has been discovered and it is valid, packets are eventually delivered from source to destination.

Property II implies that a so-called routing loop is prevented. A routing loop is a situation in which, during the route discovery process, a flawed route is formed in which nodes point to each other in a forwarding circle. Thus, packets are not delivered to the destination. Such a situation can arise if a link breaks during route discovery and a node is not notified that its route became invalid.

The conventional approach to analysing network protocols is via testing and simulation. Neither is able to rule out logical flaws in the protocols because of partial coverage of executions: simulation or test runs can often miss certain conditions dependent on timing, so called 'corner cases', thus bypassing a possible erroneous execution that may be exhibited by an implementation in future. On

the other hand, subjecting the protocol to *formal verification*, for example via *model checking*, enables detailed and exhaustive analysis of network protocols. A model of the protocol together with the required properties, usually expressed in temporal logic, is submitted to a software tool called a model checker. The process of model checking can definitively establish that the property holds, or that it is violated, in which case a trace leading to error is produced. The limitation of model checking is that only finite-state models/configurations can be handled, and thus infinite-state systems can be verified only if they have property preserving finite-state abstractions. Thus, these tools are particularly useful to demonstrate violations of properties that can serve as important feedback for the standardisation effort. A more powerful approach is that of *theorem proving*; it enables correctness proofs for all possible parameter values, but at a substantial manpower effort.

The use of formal verification methods to analyse Internet standards has been advocated in [10, 15]. A model of the protocol can be built from the standard specification and subjected to verification. In [10], a routing loop error detected in AODV version 2 with the SPIN model checker and verification of routing loop freedom was performed with the help of HOL theorem prover. However, newer versions of AODV, version 5 onwards, crucially depend on timing. The standard sets certain parameters (constants), and those are then used to assign route lifetimes and define event timeouts. There are then two issues that one needs to consider about the protocol. Firstly, properties I and II should be established with the additional proviso of "in a timely manner". More importantly, the particular combination of timing constants may have an effect on the correctness of the protocol: for example, routes may time out too early. With the exception of a small-scale study in [29], this issue has not been investigated; the models built were untimed, derived by replacing a delay with a non-deterministic timeout event. This may miss timing errors. Therefore, as already suggested in [10] [page 566], AODV from version 5 onwards necessitates a real-time verification. We address this in this paper by analysing most recent versions of AODV [6] using a state-of-the-art real-time model checker UPPAAL [24], with emphasis on how the parameters as set by the standard affect the *correctness* of routing and message delivery (properties I and II) of the protocol.

4 Modelling AODV Using Timed Automata

Since we are interested in analysing timing aspects of AODV, we model the protocol with timed automata as opposed to a C-like program in previous works [10, 15]. UPPAAL [24] is an established and widely used model checker which provides an easy to use environment for constructing timed automata models and verifying them against timed temporal logic specifications. The UPPAAL model-checking engine works on-the-fly and takes advantage of some advanced techniques to overcome the state space explosion. Experimental results show that, thanks to these techniques, UPPAAL is significantly faster than other real-time verification tools [23] and also able to verify more complex systems [7].

Some of the industrial case studies include: the *Bounded Retransmission Protocol* whose correctness was shown [16] to be dependent on correctly choosing time-out values; the *Bang & Olufsen Audio/Video Protocol*, known to be faulty, for which an error trace was uncovered [20] and a corrected model automatically verified; and the *Collision Avoidance Protocol*, which was shown to be collision free [1].

This paper concerns the latest version RFC3561-bis-01 of the standard. Every node in an AODV network acts as a sender, router (intermediate node) or destination depending on the situation. Therefore, all nodes in an ADOV network have identical functions. We first model this behaviour as a *generic* AODV node. Since an AODV network is symmetric, we can use a template of UPPAAL to simplify the model.

We define node parameters as set by the draft standard [6]. As we investigate the effect of the timing values that are suggested in the draft standard on the correctness of the protocol, we can abstract the actual control packets from the model since they are of no interest in this particular case. For simplicity, we also abstract the use of RERRs and HELLO messages since we only analyse the *route discovery* process and not the *route maintenance* process. If a node receives a data packet, and the intended destination's route is expired or does not exist, we halt the verification. The model of the generic node can be found in [14]. An *n-node* AODV network is then modelled using n instances of the generic node.

Since we have worked from the draft standard [6], the model that we have derived can serve as standard *timed specification*. We have performed an analysis of thus derived specification, and were able to confirm a routing loop error of [10] for an appropriately adapted version of the model. However, the state-space explosion means that the maximum size of the network that we could consider by direct verification is 5. For larger network sizes the verification becomes infeasible. We note, however, that the routing loop error has been exhibited in [15] with 4 nodes and in [10] with 3 nodes. We therefore seek ways to reduce the size of the model while preserving the properties of interest. For a protocol model S (*specification*) and its refinement R (*implementation*), denoted by $R \leq S$, we say R preserves S's properties of interest if $R \models \varphi$ implies $S \models \varphi$.

AODV Specification vs Implementation. Observe that, for the properties we are interested in analysing, it suffices to consider one specific sender and one destination. This can be achieved by refining the generic node into nodes that perform the specific functions, while preserving key behaviour. The generic node is thus refined separately into three main functions, the sender, destination and intermediate node, as follows:

- *sender*: this node will only generate and send RREQs, receive RREPs and send data packets,
- *intermediate node*: this node will only receive RREQs, RREPs and data packets and forward them,
- *destination*: this node will only receive RREQs and data packets, and generate and send RREPs.

Since we only consider the route discovery process, only the destination node will increment its own destination sequence number. The individual nodes in the model, see [14] for the timed automata, behave as follows:

The sender node. The *sender* will increment its *sequence number* and *broadcast_id* by one, then sends a RREQ and moves to state *wait_for_reply* to wait for a RREP. If a RREP is not received within *NET_TRAVERSAL_TIME* time (NETT), the sender times out and sends another RREQ. The NETT is doubled and the *broadcast_id* incremented by one every time a RREQ is resent. The sender will resend RREQs up to *RETRIES* times. If a RREP corresponding to a RREQ that has timed out is received, it is ignored. When the anticipated RREP is received, the sender establishes the route to the destination node and starts sending data packets. When the sender has tried *RETRIES* times for a route and still times out after the $RETRIES^{th}$ time, the sender node concludes that a route to the desired destination does not exist.

The intermediate node. The intermediate node will accept a RREQ, RREP or data packets from its neighbouring nodes, update its own routing table, broadcast the RREQ or forward the RREP/data packets to the next node along the route to the destination node for the data packets and the source node for the RREPs, according to its (intermediate node) routing table only if the route is still active. If the route has expired, the intermediate node does not forward the RREPs or data packets. If intermediate nodes are allowed to reply to RREQs, an intermediate node will generate a RREP to a RREQ if it knows the route to the destination sought. Every RREQ has a flag that is set to enable, or unset to disable, intermediate nodes to reply to RREQs.

The destination node. The destination node will accept RREQs from its neighbouring nodes, then updates its routing table, increments its *sequence number* (destination sequence number) by one, and generates a RREP. The destination node will also accept data packets. We model the destination to receive just the first data packet, and, once the first data packet gets to the destination node, we restart the route discovery process.

Establishing Refinement between AODV Specification and Implementation Models. When deriving the specialised nodes from the generic node model, we must ensure that the properties of interest are preserved through this derivation, i.e. $R \leq S$ and $R \models \varphi$ implies $S \models \varphi$. In our case, S is a parallel composition of individual components, for example $S = S_1 \| S_2$, where each component has a corresponding implementation (refinement) R_1, R_2 respectively. The principle of compositionality allows us to tackle the state space explosion in the following way.

We first need to establish that R_i are true implementations (refinements) of S_i, i.e. $R_i \leq S_i$. A number of relations are possible as refinement in the context of timed automata. Since we have used timed automata with committed locations (no delay is allowed to occur in a committed state), urgent channels (in a state where two components may synchronize on an urgent channel, no further delay is allowed) and shared variables (global variables), we work with *timed ready simulation* [21] as refinement; it relates states of one timed automaton A to

states of another timed automaton B in such away that the actions and their timings in admissible timed executions correspond (as in timed simulation) in the presence of shared variables, urgent channels and committed states. Unlike timed simulation, timed ready simulation $\leq$ is a pre-congruence for the parallel operator, that is, $R \leq S$ implies $R\|A \leq S\|A$. Let $\leq$ preserve a chosen class of properties, i.e. $R_i \leq S_i$ and $R_i \models \varphi$ implies $S_i \models \varphi$. In UPPAAL, the verification of $A \leq B$ can be reduced to the following reachability problem:

$$A \leq B \text{ iff } A\|T_B \text{ does not reach } \mathit{error}$$

where T_B is the test automaton derived from B [28]. Next, assuming we have established that $R_i \leq S_i$ holds for $i = 1, 2$, by compositionality based on the result of [2, 21] we have:

$$\frac{R_1 \leq S_1 \qquad R_2 \leq S_2}{R_1\|R_2 \;\leq\; S_1\|S_2}$$

We represent the AODV protocol *specification* as a network of generic AODV nodes, i.e. a composition of the form:

$$AODV_{spec} \equiv generic_1\| \ldots \|generic_{n-1}\|generic_n$$

and, as *implementation*, we can consider a network, where we have one sender node requesting a route, several intermediate nodes to forward packets, and one destination node, namely:

$$AODV_{impl} \equiv sender\|inter_1\| \ldots \|inter_n\|dest$$

Thus, application of the technique of [28] to the AODV node models, that is, a manual derivation of the test automaton in each case and execution of a reachability check confirming that *error* is not reached, allows us to conclude by compositionality:

$$AODV_{impl} \;\leq\; AODV_{spec}$$

In the test automaton, *error* is a designated error-location entered whenever the behaviour of $AODV_{impl}$ is outside the behaviour specified by $AODV_{spec}$. With this approach, we reduce the size of the models that have to be analysed, in a manner preserving chosen properties. We focus on *existential* properties, which are preserved under refinement, i.e. $R \leq S$ and $R \models \varphi$ implies $S \models \varphi$, where φ is of the form E<> ψ (eventually ψ) and may refer to real-time deadlines.

The model $AODV_{impl}$ is a refinement of the original specification model $AODV_{spec}$ built from generic nodes, which is nevertheless sufficiently detailed to exhibit a timing flaw in the specification, described in the next sections.

5 The Verification Approach

We consider the effect of default timing constants on the properties of eventual route formation and eventual delivery of packets. It suffices, in our case, to

assume absence of data loss. In this paper, we focus on route discovery and management and consider active routes.

A route is deemed active as long as there are data packets periodically travelling from the source to the destination along that path. Once the source stops sending data packets, the links will time out and eventually be deleted from the intermediate node routing tables. We focus on specific verification scenarios with finite static topologies and look for property violations. Since we have established refinement, it follows that any existential properties true of the implementation also hold for the specification. The model of the protocol can be investigated under different topology scenarios; the analysis we report here pertains to the (static) linear topology.

The Linear Topology Scenario. We arrange nodes of an *n-node* network into a chain, with one sender and one destination, as follows. IP addresses are selected using integers from 0 to $n-1$. Node 0 will be our originating node (sender), node $n-1$ will be the destination node, and the rest are intermediate nodes with IP addresses allocated consecutively. Thus, node 0 has node 1 as its neighbour and the destination node has $n-2$ as its neighbour.

The AODV draft standard [6] suggests that a sender tries three times to discover a route before concluding that a node cannot be reached. To allow easy instantiation of the model for different numbers n of the intermediate nodes, we have formulated an *n_nodes* node which combines n nodes linearly into one multiple node.

Thus, the obtained *n_node* has fewer states, which ensures feasibility of the verification. The correctness of the construction is confirmed by checking refinement as before. Now, we model the linear topology scenario with one sender, one destination and three identical sets of intermediate nodes, one for each RREQ attempt. In other words, we have three copies of each intermediate node running in parallel, as illustrated in Figure 1.

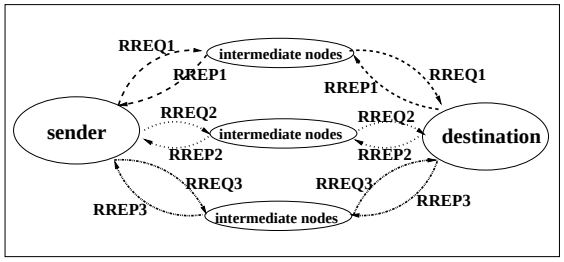

Fig. 1. The AODV linear topology model

For the remainder of the paper we consider the simplified model for $AODV_{impl}$ which employs the *n_node*. We successfully proved that twelve intermediate nodes simulate a 12*_node* multiple node. The automata models can be found in [14].

6 A Timing Analysis of AODV

In this paper we consider some of the default constants suggested in the latest version of AODV draft standard [6]. We are particularly interested in the *NET_DIAMETER* (ND) value. This value is a measure of the network size, defined as the maximum possible number of hops between two nodes in the network, and is used to determine route lifetime and time-out values. The protocol suggests that the value of ND be a constant, but does not mention how this value should be adjusted to suit the dynamic changes in network size. The following are some of the default constants suggested in [6]:

NODE_TRAVERSAL_TIME	$= 10ms$. This is the time taken by a node on average to process a packet.
NET_DIAMETER	= 20. We use 2 for model reduction.
NET_TRAVERSAL_TIME	= *2 * NODE_TRAVERSAL_TIME * NET_DIAMETER* ms. Time a sender waits for a RREP.
ACTIVE_ROUTE_TIMEOUT (abbrev. *ART*)	= $max(3000, PATH\_DISCOVERY\_TIME)$ where *PATH_DISCOVERY_TIME* = 2* *2 * NET_TRAVERSAL_TIME*. This is lifetime of a valid route. If maximum is 3000ms then we have the situation described version 13, else version 00.
RETRIES	= 3.

Though we work with average message delivery times, it is also possible to rerun the analysis for an interval of values. We separately consider two cases set out in the standard:

1. when intermediate nodes are allowed to respond to RREQs, and
2. when intermediate nodes are **not** allowed to respond to RREQs, with only the destination node allowed to respond.

Below we describe the outcome of our analysis when intermediate nodes are allowed to reply to a RREQ; the other case is omitted for reasons of space. Assume we have a linear topology with fourteen nodes in the network, one sender (number 0), one destination (number 13) and twelve intermediate nodes (numbered 1 to 12). We also assume that we have no message loss and no delays in the network. The sender (node 0) sends a message to node 13, and is allowed to try three times for a route to a destination. We refer to each route discovery attempt as the *Route Request Process* (RRP). We ensure that the sender issues five RRPs, each with three attempts, before it can conclude that a route does not exists. In real life a sender might try one RRP and conclude that a route does not exist. At a later stage the same node may try again to find a route to the same destination, maybe for a different set of data packets, and the topology might have changed.

We first investigate the correctness property I, i.e. eventual route discovery, assuming the route exists, in negated form.

I. Can a sender fail to find a route to a destination when the route exists? Using UPPPAL we verify the property 'eventually the sender reaches a state with no route found':

$$\text{E<>}\ \text{sender.no\_route}$$

where the *no_route* state is a state in which the sender learns that no route exists, for a situation in which the route is known to exist.

Because the topology remains static, the number of RREQs the sender issues does not affect the outcome. This undesirable property is satisfied and the following trace is produced, where we use 'ip[n]=m' to denote that the node with the IP address n is sending to node with address m, and similarly for destination IP address 'dip[n]=m':

RREQ trace	: $ip[0] = 13$	$ip[1] = 11$	$ip[2] = 7$
RREP trace	: $dip[0] = 1$	$dip[1] = 2$	$dip[2] = 3$

This trace (see detailed illustration in Figures 2 and 3) means that the first RREQ (RREQ1) gets to the destination node (node 13). The first RREP (RREP1) is generated and sent back to the sender. When RREQ1 is at node 4, the sender's RREQ timer times out and the second RREQ (RREQ2) is sent. RREQ2 gets to node 11 and finds a route (set by RREP1). RREP2 is generated by node 11. The sender's RREQ timer times out again and another RREQ (RREQ3) is sent. RREQ3 gets to node 7 and finds that node 7 has a route to the destination (set by RREP1). RREP3 is generated by node 7. When RREP1 gets to node 1, the route to the sender has timed out and is not forwarded. When RREP2 and RREP3 get to node 1, they are both not forwarded as well, as the route to the sender node has since expired. Thus, eventually the sender times out, failing after 3 attempts to find a route to the destination when, in fact, the route existed.

Note that, in Figures 2 and 3, we show route timers for the intermediate nodes only, i.e. nodes 1 to node 12. After 70ms, RREQ1 is at the destination and RREQ2 is at node 8. In the first step, after 40ms in Figure 2, the sender times out as the RREQ timer is initially set to 40ms. A second RREQ is sent, and the sender's RREQ timer is set to 80ms (2*40ms).

In step 3, in Figure 2, RREP1 has been generated by the destination and has been propagated to node 10. RREQ2 is at node 11. Node 11 has a route to the destination set by RREP1 and because intermediate nodes are allowed to reply to RREQs, RREP2 is generated by node 11. RREQ2 has been updating the life of the route to the sender on its way to the destination. RREQ3 is at node 2.

After 60ms, see Figure 3, RREP1 is at node 5, RREP2 at node 6 and RREQ3 at node 7. Node 7 has a route to the destination set by RREP1, and hence can reply to RREQ3. RREP3 is generated. RREQ3 has been updating the route to the sender. After 100ms, as shown in Figure 3 (since RREQ3 was sent), RREP1 is at node 1, and RREP2 and RREP3 are at nodes 2 and 3 respectively.

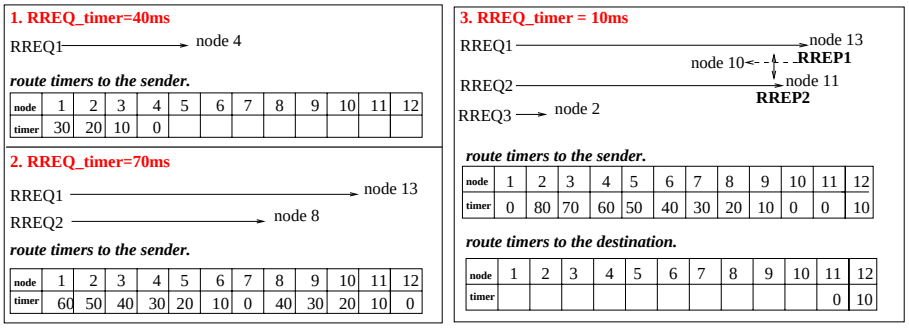

Fig. 2. RREQ1, RREQ2, RREP1 and RREP2 generations

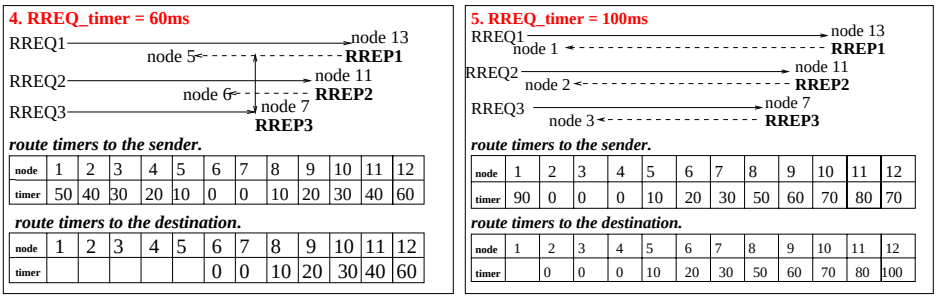

Fig. 3. RREP3 generations

The route lifetime at node 1 is 90ms (which is greater than ART=80ms), and hence RREP1 will not be forwarded to the sender. RREP2 and RREP3 will be forwarded to node 1 where the route has expired and will not be forwarded to the sender either. The sender will eventually time out and conclude that the route does not exist.

Next we consider property II, again in negated form.

II. Can a route expire before a data packet is transmitted? We assume that the sender starts sending data packets as soon as the route is found. We consider the first data packet along the way from the sender to the destination and test to see if any of the intermediate node's routes to the destination will time out before they have actually forwarded the first data packet. The property, 'eventually the intermediate node times out', is as follows:

$$\text{E<>}\ (\text{inter.data\_route\_timeout})$$

Assume we have 7 nodes in the network, one sender, one destination and five intermediate nodes. When the first data packet gets to node 5, the route to the destination at that node would have timed out, and this property is satisfied. Below is the trace that is obtained:

RREQ trace	: $ip[0] = 6 \quad ip[1] = 6 \quad ip[2] = 0$	
RREP trace	: $dip[0] = 0 \quad dip[1] = 0 \quad dip[2] = 0$	
Route Lifetime	: $route[0] = [170, inf]$	$route[1] = 30 \quad route[2] = 20$
	$route[3] = 10$	$route[4] = 0 \quad route[5] = 100$
Data trace	: $dataip = 5$	

To explain this outcome, let us consider the point when the sender has just received RREP2. Below are the values of route timers along the way to the destination:

Route Lifetime	: $route[0] = [170, inf]$	$route[1] = 0 \quad route[2] = 10$
	$route[3] = 20$	$route[4] = 30 \quad route[5] = 50$

The sender will take another 10ms to process the RREP and start sending data packets, then another 40 ms to get to node 5. Thus, by the time the first data packet gets to node 5, after 50ms the route timer at node 5 will be 100ms, which is greater than ART=80ms.

In summary, we have established that, in a network with (constant) default parameters set by the standard, the correctness requirements I and II are not satisfied. We have exhibited this property in the refined implementation model, assuming absence of message loss and delays; since it is existential, it follows that it is also exhibited by the specification, and in a realistic scenario with message loss and delays.

How to define NET_DIAMETER. The AODV draft standard may be improved by allowing the value of *NET_DIAMETER* to grow with the network size. Initially, we can set it to be e.g. the constant suggested by the standard, and then let individual nodes modify it. If a node receives a RREQ, RREP, or RERR packet that has a hop count that is greater than the node's *NET_DIAMETER*, then the node should adjust its *NET_DIAMETER* to the value of this hop count. This allows the nodes to learn and adapt to the new network size.

We have modified the model accordingly and re-verified the corrected model for the variant where intermediate nodes are allowed to reply to RREQs. The automata for this model can again be found in [14]. We observe that the properties I and II are now satisfied; in particular, the validity of routes is prolonged. Note that this does not amount to a full verification of correctness, which would have required a theorem prover, but is automatic.

For a decrease in network size, we propose to leave the *NET_DIAMETER* unchanged. This ensures that, if a route exists, the requesting node will eventually find it. However, if the route does not exist, then the requesting node has to wait longer before it can conclude that this is so.

7 Related work

Model checking has been successfully used to analyse various distributed protocols, but few papers have applied model checking in the context of mobile ad

hoc network routing. We mention the discovery of a routing loop error found in early versions of AODV with the SPIN model checker [8] and Murphi [15]. Engler et al [18] have analysed three AODV implementations using CMC (a model checker for the C programming language), reporting several errors of which one can be attributed to the standard specification on RERR handling. In their earlier work [15], they explain how reordering of RERR messages by the link layer could lead to a routing loop. Other errors reported include mishandling of memory allocation, missed essential checks of packets and routing loops. Their handling of time to ensure a compact model can miss timing errors. A predicate abstraction approach (requiring human intervention) was used in [17] to verify the absence of routing loops. In [9], an automated proof of routing loop freedom is given under certain conditions, using the model checker SPIN and theorem prover HOL. Real-valued time-outs are represented as non-deterministic time-out events, which does not faithfully model real-time passage.

None of the verification studies that we are aware of considered the timing aspects of AODV, with the exception of [29] which used DTSpin [11], an extension of SPIN with discrete time, but was not as extensive as our study. They reported that their DTSpin version was too unstable and abandoned their automatic verification attempt using DTSpin in favour of a manual proof. All the above AODV studies concern earlier, less complex versions of the standard. Some aspects of timing properties have been analysed for the LUNAR protocol [27], using the UPPAAL model checker, but for very small models only.

The *NET_DIAMETER* issue has been raised briefly on the MANET mailing list [26, 25], but not followed up since. Preliminary reports of this work appeared as [13, 14].

8 Conclusion

We have modelled the AODV protocol with timed automata and analysed certain configurations using a number of techniques developed for the UPPAAL model checker to obtain model reductions. We observe that the protocol as specified may unnecessarily result in failure to discover the route or deliver the message. The problem occurs because nodes wait for a fixed time for RREPs that, in a dynamically growing network, may take much longer to reach the requesting node. We propose a modification to the AODV routing algorithm by allowing the nodes to amend the value of *NET_DIAMETER* through learning about the size of the network from the incoming packets. To our knowledge, this is the first solution to this problem.

In contrast with previous work, we have analysed the latest draft specification [6], as well as earlier versions 00 [4] and 13 [5]. All three exhibit this problem, albeit in slightly different form. As a sanity check, we ran ns-2 experiments for the AODV-UU implementation code [30] that complies with version 13, confirming our observations for this implementation also by obtaining identical traces to those exhibited by the model. We have notified the AODV authors about our findings and they have accepted our suggestions [3].

References

1. L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The Power of Reachability Testing for Timed Automata. In *Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science, Chennai, India, December 1998*, volume 1530, pages 245–256. Springer, December 1998.
2. L. Aceto, A. Burgnueno, and K. G. Larsen. Model Checking via Reachability Testing for Timed Automata. Technical Report RS-97-29, BRICS, November 1997.
3. E. M. Belding-Royer and I. D. Chakeres. Private e-mail communication.
4. E. M. Belding-Royer, I. D. Chakeres, and C. E. Perkins. Ad hoc On-Demand Distance Vector (AODV) Routing. *Work in progress*, September 2003. http://www.ietf.org/internet-drafts/draft-perkins-manet-aodvbis-00.txt.
5. E. M. Belding-Royer, I. D. Chakeres, and C. E. Perkins. Ad hoc On-Demand Distance Vector (AODV) Routing. *Work in progress*, February 2003. http://moment.cs.ucsb.edu/AODV/ID/draft-ietf-manet-aodv-13.txt.
6. E. M. Belding-Royer, I. D. Chakeres, and C. E. Perkins. Ad hoc On-Demand Distance Vector (AODV) Routing. *Work in progress*, July 2004. Internet Draft, RFC 3561bis-01, http://moment.cs.ucsb.edu/pub/draft-perkins-manet-aodvbis-02.txt.
7. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - A Tool Suite for Symbolic and Compositional Verification of Real-Time Systems. In *Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*. Springer–Verlag, May 1995.
8. K. Bhargavan, C. A. Gunter, M. Kim, I. Lee, D. Obradovic, O. Sokolsky, and M. Viswanathan. Verism: Formal Analysis of Network Simulations. In *Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2000)*, pages 2–13, Department of Computer Science, Stanford University, Stanford, CA 94305, U.S.A, August 2000.
9. K. Bhargavan, C. A. Gunter, I. Lee, O. Sokolsky, M. Kim, D. Obradovic, and M. Viswanathan. Verisim: Formal Analysis of Network Simulations. *IEEE Transactions on Software Engineering (TSE)*, 28(2):129–145, 2002.
10. K. Bhargavan, C. A. Gunter, and D. Obradovic. Formal Verification of Standards for Distance Vector Routing Protocols. *Jounal of the ACM*, 49(4):538–576, July 2002.
11. D. Bosnacki and D. Dams. Discrete Time Promela and SPIN. Dept. of Computing Science, Eindhoven University of Technology ,Eindhoven, The Netherlands. DTSpin web site: http://www.win.tue.nl/ dragan/DTSpin/.
12. I. D. Chakeres and E. M. Belding-Royer. AODV routing protocol implementation design. In *Proceedings of the International Workshop on Wireless Ad hoc Networking (WWAN)*, pages 698–703, Tokyo, Japan, March 2004.
13. S. Chiyangwa and M. Z. Kwiatkowska. Modelling Ad hoc On-Demand Distance Vector (AODV) Protocol with Timed Automata. In *Proceedings of the Third Workshop on Automated Verification of Critical Systems (AVoCS'03)*, Southampton, UK, April 2003.
14. S. Chiyangwa and M. Z. Kwiatkowska. Analysing Timed Properties of AODV using UPPAAL. Technical Report CSR-04-4, School Of Computer Science, The University of Birmingham, UK, March 2004.
15. A. Chou, D. L. Dill, D. R. Engler, M. Musuvathi, and D. Park. CMC: A Pragmatic Approach to Model Checking Real Code. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

16. P. R. D'Argenio, T. C. R. Joost-Pieter Katoen, and G. J. Tretmans. The Bounded Retransmission Protocol Must Be on Time! Technical Report CTIT 97-03, Faculty of Computer Science, University of Twente, The Netherlands and Lehrstuhl fur Informatik VII. University of Erlangen, Germany, 1997.
17. S. Das and D. L. Dill. Counter-Example Based Predicate Discovery in Predicate Abstraction. In *Proceedings of the Fourth International Conference on Formal Methods in Computer-Aided Design (FMCAD) 2002*, volume 2517 of *Lecture Notes in Computer Science*, pages 19–32, Portland, Oregon, USA, November 2002. Springer-Verlag.
18. D. R. Engler and M. Musuvathi. Static Analysis versus Software Model Checking for Bug Finding. In *Proceedings of the Fifth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI '04)*. Springer-Verlag, 2004.
19. E. N. et al. AODV-UU: Ad-hoc On-demand Distance Vector Routing. http://user.it.uu.se/~henrikl/aodv/.
20. K. Havelund, K. G. Larsen, K. Lund, and A. Skou. Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using Uppaal . In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pages 2–13, San Francisco, California, December 1997. BRICS RS-97-31.
21. H. E. Jensen, K. G. Larsen, and A. Skou. Scaling up Uppaal - Automatic Verification of Real-Time Systems using Compositionality and Abstraction. In *Proceedings of the Sixth International School and Symposium on Formal Techniques and Fault Tolerant Systems (FTRTFT00)*, volume 1926 of *Lecture Notes in Computer Science*, pages 19–30, Pune, India, 2000. Springer-Verlag.
22. K. G. Larsen and P. Pettersson. Uppaal2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000. http://www.uppaal.com.
23. K. G. Larsen, P. Pettersson, and W. Yi. Compositional and Symbolic Model-checking of Real-time Systems. In *RTSS '95: Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, page 76. IEEE Computer Society, 1995. ISBN 0-8186-7337-0.
24. K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
25. J. Macker. Re; [manet] 400 nodes in NS. MANET mailing list, 26 July 2002. Message ID: 5.1.1.5.2.20030313133000.01565e78@pop.itd.nrl.navy.mil.
26. P. Neumiller. OSPF to AODV bridging. MANET mailing list, 13 March 2003. Message ID: 01a701c234a39027b470ab0110ac@meshnetworks.com.
27. J. Parrow, A. Pears, and O. Wibling. Automatized Verification of Ad Hoc Routing Protocols . In *Proceedings of the Formal Techniques for Networked and Distributed Systems (FORTE), FORTE'2004*, volume 3235 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
28. D. P. L. Simons and M. Stoelinga. Mechanical Verification of the IEEE 1394a Root Contention Protocol using Uppaal2k . *International Journal on Software Tools for Technology Transfer*, 3(4):469–485, 2001.
29. R. Talreja. Formal Verification of AODV. Master's thesis, Department of Computer Science and Information Science, University of Pennsylvania, April 2002. Advised by Carl Gunter and Karthikeyan Bhargavan.
30. B. Wiberg. Porting AODV-UU Implementation to ns-2 and Enabling Trace-based Simulation. Master's thesis, Department of Computer Science, Uppsala University, Sweden, December 2002.