Loop Freedom in AODVv2

Kedar Namjoshi, Richard Trefler

Loop Freedom in AODVv2

Kedar S. Namjoshi[1] * and Richard J. Trefler[2] **

[1] Bell Laboratories, Alcatel-Lucent kedar@research.bell-labs.com
[2] University of Waterloo trefler@cs.waterloo.ca

Abstract. The AODV protocol is used to establish routes in a mobile, ad-hoc network (MANET). The protocol must operate in an adversarial environment where network connections and nodes can be added or removed at any point. While the ability to establish routes is best-effort under these conditions, the protocol is required to ensure that no routing loops are ever formed. AODVv2 is currently under development at the IETF, we focus attention on version 04. We detail two scenarios that show how routing loops may form in AODVv2 routing tables. The second scenario demonstrates a problem with the route table update performed on a `Broken` route entry. Our solution to this problem has been incorporated by the protocol designers into AODVv2, version 05. With the fix in place, we present an inductive and compositional proof showing that the corrected core protocol is loop-free for all valid configurations.

1 Introduction

The AODV ("Ad-Hoc On-Demand Distance Vector") protocol family is under development by the IETF MANET (Mobile, Ad-Hoc Networking) group. Its current form is AODVv2[3], which has evolved from the earlier DYMO[4] and AODV[5] protocols. As stated in the protocol description, AODVv2 "*is intended for use by mobile routers in wireless, multihop networks. AODVv2 determines unicast routes among AODVv2 routers within the network in an on-demand fashion, offering rapid convergence in dynamic topologies.*" AODVv2 is still evolving; our work focuses on the recent version 04 (which we refer to as AODVv2-04), published in July 2014. Subsequently, AODVv2-05 was issued in October of 2014, and AODVv2-06 in December 2014.

* Supported, in part, by DARPA under agreement number FA8750-12-C-0166. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

** Supported in part by Natural Sciences and Engineering Research Council of Canada Discovery and Collaborative Research and Development Grants.

[3] http://datatracker.ietf.org/doc/draft-ietf-manet-aodvv2/
[4] http://datatracker.ietf.org/doc/draft-ietf-manet-dymo/
[5] http://datatracker.ietf.org/doc/rfc3561/

The environment in which AODVv2 operates is challenging, as network connections and nodes may be added or removed at any point. In such a setting, routes are established in a best-effort mode. However, the protocol is required to enforce a key safety property, that there are no routing loops in any reachable global state. A routing loop is formed when the next-hop entries in routing tables are connected in a cyclic manner (E.g., node A has next-hop B; node B has next-hop C; and node C has next-hop A).

We construct a formal, inductive proof that an *abstract model* of the protocol has no routing loops. Such a proof has utility, even though AODVv2 is still not finalized. A proof elucidates broad conditions under which loop-freedom can be guaranteed; those conditions can then be taken into account as the protocol is refined, and any fixes necessary can be incorporated quickly and with relatively little cost. Indeed, in the course of our analysis, we found that AODVv2-04 allows routing loops to form under certain sequences of actions, we discuss those scenarios in Section 1.2. The first example illustrates that if instance-specific timing constants are not set correctly, then routing loops may form. The second example is more serious since it can occur even if the protocol parameters are set correctly. This problem was quickly acknowledged by the protocol designers and corrected, based on our input, for version 05 of AODVv2 (cf. [14], `Appendix C: Changes since revision ...-04.txt`).

Our model aims to capture the core of the AODVv2 protocol by abstracting away some detail and by leaving out optional features. The main abstraction is that timer-driven actions are replaced either with non-determinism or with global predicate guards. For instance, our model allows a route entry to be invalidated at any point, while the protocol permits this only after timer expiration. In another instance, routes marked as `Expired` are expunged (i.e., removed completely) only after there is no activity for at least `MAX_SEQNUM_LIFETIME` seconds. Note that `MAX_SEQNUM_LIFETIME` is one of several instance specific constants in the protocol. Our model abstracts away from such constants by replacing the time-based preconditions with global network predicates. The predicates abstract nicely from the specifics of network structure, delays and processing power, which must go into determining a correct setting for these symbolic constants.

The full AODVv2 protocol has mechanisms that allow a wide variety of metrics to determine the cost of a route. Our model considers only the hopcount metric, i.e. the metric that counts the number of network edge hops between the origin node of a route and the target node of a route. Hop count is an important metric used in practice, and AODVv2 correctness requires at a minimum that the protocol behave correctly with the hop count metric. Our proof can be adapted to other cost metrics, where the cost of a link is greater than 0, and cost along a path is additive. Within these limits, the model exhibits all of the actual protocol computations, and more: hence, any proof that the model is correct shows also that the protocol, under the stated restrictions, is correct.

To summarize, our work makes two contributions: (1) we exhibit scenarios where AODVv2-04 allows routing loops to form, and suggest a protocol fix,

which has been adopted by the designers, and (2) we construct a formal model of the protocol and an inductive proof showing that the corrected core protocol ensures loop freedom. This proof is interesting in its treatment of adversarial actions and its use of compositional reasoning.

1.1 Protocol Sketch

We informally sketch the main features of the protocol before proceeding to the proof. The model we use is given in Section 3. The model fixes an origin node, O, and a target node, T. The protocol establishes a route from O to T in two phases. The first phase is initiated by O, and consists of flooding a RREQ (route request) message through the network[6]. Every node receiving this RREQ message maintains an "origin route", next-hop entry which points to the neighboring node from which it has received the best route so far from O, i.e., (roughly) the (first) path with the least cost. Note that a node may receive multiple copies of the RREQ message sent from O, through different paths. Whenever the target node T receives a better RREQ route from O, it responds with an RREP (route reply) message. This message is not flooded: it follows (backwards) the path to O that has been established by the origin route entries. With fixed network connectivity and no message losses, this procedure converges (under mild conditions) to a least cost path from O to T. Under the network disruptions that are expected in the MANET model, though, there is no guarantee of convergence. Under adversarial control of the network and message transmission, the only property that is *required* of the protocol is that it should never form a global state which has a routing loop: i.e., a state where the set of origin route entries form a cycle, such as where node A has next-hop B, B has next-hop C, and C has next-hop A.

The tricky part of the analysis has to do with the case of "broken" route entries, which are created when links in the network fail. If A has next-hop B and the $A-B$ link fails, then the entry at A is marked as `Broken`. However, new copies of the RREQ message from O may arrive at A after the breakage. *When should a route from one of those messages be accepted at A?* Accepting any route at all – which makes sense in a way: an unbroken route, however bad, is surely preferable to a broken one – may lead to a routing loop, as shown in the second scenario below. This scenario was possible in version 04 of the protocol, and it was discovered by us in the attempt to construct a proof of loop-freedom[7]. The

[6] A data structure, the `RREQ Table`, is used in AODVv2-04 to control the flooding. Appendices A.1 and A.2.2 of AODVv2-04 describe precisely how the table is used: an incoming `RREQ` message is used to update a route entry, then the message is checked against the table to determine if it should be regenerated and sent to neighboring nodes. (We have confirmed this order of actions with the protocol authors, to resolve a slight ambiguity in the main text.) Hence, the table does not influence route updates; it may only stop the regeneration of `RREQ`s, which is already included in our model as message loss. Therefore, we do not model the table.

[7] From Section 6.3 of AODVv2-04, one case of the condition for acceptance of a new route is "`((Route.State == Broken) && LoopFree(RteMsg, Route))`". The

partial proof pointed to the condition "accept any route that is not worse than the current broken route" as a possible resolution. We confirm that this is indeed a correct resolution through the formal proof given next. That resolution has also been accepted by the authors of AODVv2 and included starting with revision 05 of AODVv2 (cf. [14], `Appendix C: Changes since revision ...-04.txt`).

1.2 Loop Formation Scenarios

The first scenario creates a loop when the timer `MAX_SEQNUM_LIFETIME` is not set to a large enough value. The second creates a loop when any route is accepted in place of a broken one. Reading through the scenarios helps build intuition about how the protocol operates, which is helpful in understanding the proofs.

Poor choice of timer values. The AODVv2 protocol has several actions that are triggers by symbolic time constants. A protocol implementer has to give concrete values to these constants, a very difficult decision, as the correct values depend on the topology of the network, processing speeds, and transmission delays. As shown in the scenario below, a routing loop may result if the constants are inadvertently not set properly. The loop prevents RREP messages that are sent back from T, the target node, from reaching O, the originating node. As a result, no messages can be transferred from O to T.

We should note that this is *not* an error in the protocol – with a correct choice of constants, the loop will not occur. We model the time-based actions as guarded commands with an untimed guard over the network state. The proof shows that the model is loop-free. The modeling, therefore, helps to narrow down the choice of time constants: the values chosen for a network instance should be such that the guard condition is guaranteed to hold when the timers expire.

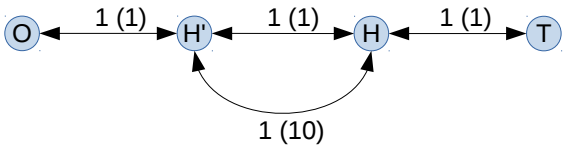

Fig. 1. Network: Early Expunge Scenario. Number by edge indicates hop-count, number in brackets indicates transmission delay in time units.

The network is fixed as shown in Figure 1. The scenario is as follows:

predicate `LoopFree` is defined in Section 5.6 as "`LoopFree (R1, R2) is TRUE when Cost(R2) <= (Cost(R1) + 1)`". Thus, `LoopFree(RteMsg,Route)` is true iff `Cost(Route) <= (Cost(RteMsg) + 1)`. This allows the cost of the route in the incoming message, `RteMsg`, to be arbitrarily larger than the cost of the stored route, `Route`, if the stored route is in a `Broken` state.

1. An RREQ message created by O travels along the path O; H'; H; T. As a result, the origin route entries at these nodes have hop counts O=0, H'=1, H=2, T=3. A copy of the RREQ message remains undelivered on the link $H - H'$.
2. The route entry at H' is expired and then expunged.
   This is the critical step. In AODVv2-04, timer conditions say when a route *must* be expunged. For non-timed routes, this happens (ref. Section 6.3) when `(Current_Time - Route.LastUsed)>= MAX_SEQNUM_LIFETIME`. However, the protocol (ref. Section 6.3) also allows routes to be expunged without reference to `MAX_SEQNUM_LIFETIME`: an Expired route *may* be expunged at any time (least recently used first). If this constant is set to too low a value, there will be messages within the network which are still undelivered. In the network of Figure 1, if `MAX_SEQNUM_LIFETIME` is set to 4 units, and the $H - H'$ path (a single link is shown but it could be a path through intermediate nodes) has the delay shown in the figure, the protocol will force the routing entry at H to be expunged while there is an undelivered RREQ. The correct value depends on many factors, including the size of the network, the length of paths in the network, and processing speeds and buffering at the nodes. In the model, we abstract this to a global predicate which must be met before the expunge action can occur.
3. The undelivered RREQ from H now reaches H'. Since H' has no entry, it accepts this route; its next-hop is now H.
4. H' sends a RREQ to H with hopcount 3. Since H already has an entry with a better hopcount, it rejects this message. At this point, there are no undelivered RREQ messages. The H'-H entries form a routing loop.

Broken Routes. A route entry at a node is marked as `Broken` if the node is made aware of a break in connectivity. The following scenario shows that a loop may form if a *broken* route is replaced by any *valid* route (as may seem reasonable, even if the new route has a higher cost).

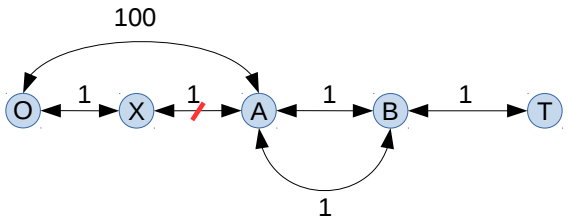

Fig. 2. Network: Broken Route Scenario. Number over edge indicates hop-count (cost); red (slanted) line indicates break.

1. A RREQ message generated at O sets up the route O(0);X(1);A(2);B(3);T(4). The numbers in () are the hopcounts for the origin route entries at each node. A RREQ message remains undelivered on the lower A-B link.

2. The link X-A breaks, causing the route at A to be marked as `Broken`. After the link breaks, both X and A are required to send out RERR messages to their neighbors. We assume that those messages are lost and therefore neither O nor B is notified of the break.
3. A now receives a long route from O, with cost 100. This is the critical point. In version 04 of AODVv2, any valid route is acceptable in place of a broken one (see footnote 7 for details on why this is permitted), so this route will be accepted. The route entry at A is now valid and has hopcount 100.
4. A now has a non-broken route. It receives the previously undelivered RREQ from B, which has cost 4. As this cost is less than that of its current route cost (i.e., 100), A switches its next-hop to B. Node A then sends an RREQ to B, but that has higher cost than B's current route, and is rejected. No further RREQ messages remain in the network, so the A-B loop is stable.

2 Proof of Loop Freedom

The proof method is standard: we identify a suitable assertion and prove that it is an inductive invariant by showing that it is preserved by every action. However, the proof structure is more interesting: (1) we explicitly model network disruptions as adversarial actions and (2) the induction proof is localized to the neighborhood of an arbitrarily chosen network edge; thus, it implicitly uses symmetry and is compositional in nature. Some aspects of the model are especially important for the proof (see Section 3 for more details of the model):

1. There is an underlying connectivity graph of nodes. We assume that the graph is finite but of arbitrary size. Nodes and links may fail, and new links can be formed at any point. A node can also be restarted after failure.
2. Any link change and the reaction to it happens atomically with respect to the actions of the protocol.
3. We fix an arbitrary origin node, O, and an arbitrary target node, T, such that T differs from O. Protocol analysis is then based on the discovery and maintenance of bidirectional routes from O to T.
4. A route entry has a sequence number, a hop-count, and a state[8]. We say that an entry x is "better" than an entry y if $(seq_x, -hop_x)$ is lexicographically strictly greater than $(seq_y, -hop_y)$. I.e., if $seq_x > seq_y$ or if $seq_x = seq_y$ and $hop_x < hop_y$. In this situation, we also say that y is "worse" than x. We write this relationship as $y \prec x$. We treat sequence numbers as natural numbers; i.e., we do not model wrap-around effects. In AODV-v04, a node has its own sequence number generator, with the range $[0 \ldots 65535]$. A new sequence number is assigned for a fresh route request/response. As the numbers are assigned per node and the protocol separates routing entries by (origin, target), the AODVv2 drafts implicitly assume that comparison of a route with a wrap-around successor route is very unlikely.

[8] In AODVv2-04, an entry is also labeled with an (origin, target) pair. As the model fixes the origin and target nodes, we omit this label.

2.1 Proof Summary

We consider first the origin route established through RREQ messages, and show that it cannot have a routing loop. To avoid case-splitting, we suppose that there is a dummy route to O at O, given by hopcount 0 and the sequence number of O. The proof hinges on showing the following lemma, which gives the desired theorem below.

Lemma 1 *The following is an inductive invariant: for any node H, and for any node G: if H has a route entry to O with next hop G, then G has a route entry to O that is* ***better than*** *the entry at H.*

Theorem 1 *The protocol never reaches a state with a routing loop formed from origin route entries.*

Proof: The proof is by contradiction. Suppose that there is a reachable protocol state with a routing loop induced by the entries for origin routes. Pick a node, say H, on the loop other than O (there must be one such) and go around the loop from H in the next-hop direction. By Lemma 1, the route entries along this circuit improve strictly at each hop. By the transitivity of $\prec$, the route entry at H is strictly worse than the route entry at H, a contradiction. **EndProof.**

2.2 The Main Proof

The bulk of the proof lies in establishing the invariance condition in Lemma 1. We do so by induction: i.e., we show that the statement holds in the initial protocol configuration, and that it is preserved by protocol actions and by dynamic network changes. For brevity, we use "route" in place of "route entry" throughout; it should be understood that route does not refer to a path connecting several nodes together. We require an auxiliary lemma, given below. It states that routes in RREQ/RREP messages on outgoing channels adjacent to a node are no better than the corresponding route at that node.

Lemma 2 *The following is an inductive invariant:*

(a) *For any node H, the route to O in any RREQ message for (O,T) on any outgoing link from H is not better than the route for O at H.*
(b) *For any node H, the route to T in any RREP message for (O,T) on the link from H to its next-hop on the route to O is not better than the route for T at H.*

Proof of Lemma 2(a): The claim holds trivially at the initial state, as all connection links are empty.

Consider a transition from global state s to global state t, and suppose that the claim holds at s. To show that it holds for t, consider a node H (other than O) in t. The proof is by case analysis on the transition which takes the system from s to t.

Consider first the normal operations. Most cases are straightforward. If the transition is for a node other than H, it only affects the neighborhood of H if a message is removed from an outgoing link of H; in this case, the invariant is trivially preserved. Changes to the route state of H (idle-route or expire-route) do not change any routes. Expunging the route (expunge-route) preserves the invariant as its guard requires all outgoing channels from H to be empty. The generation of a RREQ (rreq-gen) can only be done by H if it is O: in that case the route generated is equal to the (dummy) route for O at O.

The interesting case is where the route at H is updated through a RREQ or RREP message (rreq-recv, rrep-recv). Let x be the route at H to O in s and let y be its route in t. Then either $y \succ x$ (in the normal case) or $y \succeq x$ (if x is `Broken`). Now for every route r in a RREQ message on the link in s, the inductive hypotheses requires that $x \succeq r$, so that $y \succeq r$ by transitivity. Every new RREQ message generated by H through rreq-recv carries the route y. This re-establishes the invariant. Processing an RERR message may only invalidate but not change the origin route.

We now consider the dynamic changes. Dropping a message, and removing a node or a link trivially preserves the invariant as no routes are changed. The addition of a link to H establishes the invariant for that link, as the link is empty. The interesting case is if H is a recovered node (recover-node). By the pre-condition for recovery (see the model detailed in the next sections), all of H's outgoing channels are empty, so the invariant holds. **EndProof.**

The proof for part (b) is essentially identical, as the processing of RREQ messages in rreq-recv and RREP messages in rrep-recv is nearly symmetric.

Proof of Lemma 1: The claim holds trivially at the initial state, as all routes are undefined. Consider a transition from global state s to global state t, and suppose that the claim holds at s. We show that it holds at t by case analysis on the transition.

We first consider the normal protocol actions. In state t, consider node H, and a node G such that the route to O from H has next-hop G. We have to show that G has a route better than the route at H. Consider the possible actions.

(1) The action does not involve either H or G. So there is no change in the routes at the two nodes. By assumption, the claim holds for (G, H) in s, so it continues to hold in t.

(2) The action is one of G. Modifications to route state (idle-route, expire-route) do not affect routes, so the claim continues to hold from the assumption for s. The action cannot be an expunge, as its guard is not met in s, as the entry for H in s has next-hop G. Processing of RERR messages does not change the route at G (although its state may change). The interesting case is where G updates its route to O from r_G in s to r'_G in t by processing an RREQ or an RREP message. By the protocol, $r'_G \succeq r_G$. Since $r'_H = r_H$, and $r_G \succ r_H$ by assumption for s, we get that $r'_G \succ r'_H$ in t.

(3) The action is one of H. Modifications to route state (idle-route, expire-route) do not affect routes, so the invariant is preserved from s. The action cannot be an expunge, as H has an entry in t. The interesting case is if H

updates its route to O through a rreq-recv for (O, T) or through a rrep-recv for (X, O), where X is some node. Since H points to G in t, the updating message must be from G. Say this message carries a route r, and let r_G be the route to O at G in s. By Lemma 2, regardless of the message type (RREQ or RREP), at state s, $r_G \succeq r$. The new route at H is obtained from r by incrementing its hopcount, so it is worse than r (i.e., $r'_H \prec r$). The route in G is unchanged in the transition (i.e., $r'_G = r_G$). Hence, we have $r'_G = r_G \succeq r \succ r'_H$. By transitivity, $r'_G \succ r'_H$, as is desired. (Note the crucial role played by the hopcount increment at H.) Actions which process RERR messages do not change the route at H, so they preserve the invariant.

We now consider dynamic changes which affect H and G.

(4) Dropping a message from a link, and removing a link trivially preserves the invariant as no routes are changed. (Note that the link between H and G may be broken by the transition, yet H's route entry still points to G in t.)

(5) The action cannot be the addition or restart of H, as the newly added H would not have an origin route in t. The action may not add G as a fresh node either, as the next-hop entry for G exists for H in s.

(6) The action cannot be the restart of G, as its precondition requires there to be no entries which have G as a next-hop, and H has such an entry in s.

(7) The action cannot be the removal of nodes G or H, as we are only stating the claim where both nodes exist in the network at t. (In t, there may be a node H' which has a next-hop entry for G', but G' is no longer in the network at t. Such a (G', H') pair is not part of the invariant claim.) **EndProof.**

RREP Invariants RREP (route response) messages are generated whenever a new RREQ message reaches its target. They follow *a single* path from target to source which is set up by the origin route entries. I.e., unlike RREQs, the RREP messages do not flood the network. The RREP messages create "target route" entries at each node, which determine a path from that node to the target, T. However, the origin route path at the point an RREP message is created may change as the protocol progresses and intermediate nodes receive better routes. It may also change as the result of network disruptions and rearrangements. Hence, it is not obvious that RREP messages do not induce a routing loop in the target route entries. The proof that the target routes created by RREP messages is loop-free is similar in structure to the RREP loop-freedom proof. This is possible as the protocol is nearly symmetric in its handling of RREQ and RREP messages. We therefore omit this proof.

3 AODVv2 Model

We describe the protocol model from the viewpoint of a node with name H.

Data Structures. A node maintains a route table route, indexed by nodes. The route to a node may be undefined, which we denote by $\perp$. If defined, a route to a node is a pair: (n, e), where n is the next-hop node and e is its route entry. An

entry is of the form (s, h, x), where s is a sequence number, h is the hopcount (or, more generally, the cost), and x is the state of the route (one of Active, Idle, Expired, or Broken). It is assumed that s and h are non-negative numbers. In addition, a node maintains its own sequence number, referred to as seq. We use standard notation to refer to these components, for instance, $n.\mathsf{route}[O].e.h$ refers to the hopcount of the route entry to node O at node n.

Messages. The protocol has three types of messages: $RREQ$ (route request), $RREP$ (route reply) and $RERR$ (route error). Each message has the following components: h (a hopcount), $tlv = (sO, sT)$ (sequence numbers for origin and target, possibly undefined), and (O, T) – the origin and target pair. We write a message as, for example, $RREQ(h, (sO, sT), (O, T))$.

Initial State. In its initial state, a node has undefined origin and target routes, and sequence number 0.

Protocol Actions. Here, we list the actions taken during normal operation. The actions are atomic but may occur at any time. In the protocol, actions such as expire-route are based on timers, to ensure that they do not happen too often. Since we are concerned with correctness, not performance, we replace such uses of timing by non-determinism. There are some parts of the protocol where timed actions are used as a proxy for global conditions. In the model, we replace such timers with global guards.

In the description below, we have also made certain actions (e.g., processing of RERRs) have more effect, or be more often enabled, than the actual protocol recommends. This can only result in the model having more executions than the actual protocol, so any invariants shown for the model also hold for the protocol.

The notation $y >> x$ expresses that the route in the route message y is preferable to the route table entry x. From the AODVv2 protocol description, this is true if (1) $y.s > x.s$, or if (2) $y.s = x.s$, and either (a) $y.h+1 < x.h$, or (b) x is in the Broken state and $y.h+1 \leq x.h$. (Term (b) is the correction introduced in AODVv2-05 based on the second loop-formation scenario from Section 1.2.)

We introduce the global predicate $\mathsf{AllClear}$, which replaces the time-driven actions based on `MAX_SEQNUM_LIFETIME`. The predicate $\mathsf{AllClear}(H)$ holds iff (1) there are no messages in any channel of the network with origin or target being H, and (2) all outgoing channels from H are empty, and (3) no other node has an Active route entry with next-hop H. This global condition is not present in the actual protocol, as it cannot be checked locally. The protocol instead defines a symbolic time constant, `MAX_SEQNUM_LIFETIME` – a node waits until that much time has expired before expunging an entry. The protocol description does not specify how this value is to be chosen for a network instance: the value should, clearly, depend on factors such as the size of the network, the link delays, and the processing power of a node. The global condition defined here abstracts from these considerations: the time value should be set so that the global condition is guaranteed to be true after that much time has elapsed.

skip do nothing

expunge-route remove route if its state is Expired, and AllClear(H) holds.
idle-route change route state to Idle if Active.
expire-route change route state to Expired if Idle.
rreq-gen(T) This generates an RREQ (request) message to node T.

```
true ==>
      let msg = RREQ(h=0, (sO=H.seq+1, sT=H.route[T].e.s), (H,T)) in
      H.seq := H.seq+1;
      multicast(msg)
```

rreq-recv(RREQ(m),K) This action processes an RREQ message $m = (h, (sO, sT), (O, T))$ from neighbor K. It is guarded by the condition that the route in m is better than the origin route at node H.

```
(m.sO,m.h,Active) >> H.route[O].e ==> // m has a better route to the origin
      // update the origin route
      H.route[O] := (K,(m.sO,m.h+1),Active);

      // propagate or reply as appropriate
      if (H=T) then // H is the target node: reply with RREP
         let reply = RREP(h=0,(sO=m.sO,sT=H.seq+1), (O,T)) in
         H.seq := H.seq+1; // update local sequence number
         unicast(reply, K) // send only to K
      else // H is an intermediate node: propagate
         let msg = RREQ(m.h+1, m.tlv,(O,T)) in
         multicast(msg)  // send to all neighbors
      endif
```

rrep-recv(RREP(m),K) This action processes a reply (RREP) message $m = (h, (sO, sT), (O, T))$ from neighbor K if it contains a better target route.

```
(m.sT,m.h,Active) >> H.route[T].e ==>  // m has better route to the target
      // update the target route
      H.route[T] := (K,(m.sT,m.h+1),Active);

      // propagate as appropriate
      if (H = O) then // H is the origin node: do nothing
         skip
      else // H is an intermediate node
         if (H.route[O] is defined) then // propagate RREP
            let replymsg = RREP(m.h+1, m.tlv, (O,T)) in
            unicast(replymsg, H.route[O].n)
         else // generate error RERR
            let errormsg = RERR(h=0,tlv=(_,_)) in
            unicast(errormsg,K)
      endif
```

rerr-recv(RERR(m),K) This action processes an error (RERR) message from neighbor K. Mark any routes passing through K as broken, and propagate the error. This is more permissive than the protocol in marking routes as Broken: in the protocol, there are other fields in the RERR message which H can use to distinguish whether the error message from K pertains to an origin or a target route.

```
true ==>
      for all nodes w:
```

```
        if (H.route[w].n = K) then
          H.route[w].e.x := Broken; // mark route as broken.
          multicast(RERR(m))    // propagate RERR to all neighbors
      endif
```

Dynamic Actions. We now describe protocol actions taken in response to dynamic changes. In our model the adversary may add, recover, or delete nodes, and may add or delete edges. Edges may be deleted by the adversary at any point during protocol execution. However, the adversary may delete a node only if the node is not linked to any edge. Below we give the detailed response that the protocol takes to adversarial actions.

remove-node(H) Do nothing.

new-node(H) If H is a new node, it starts at its initial state, and all outgoing channels are empty.

recover-node(H) H is a recovered node. It does not re-join the protocol until the condition $\mathsf{AllClear}(H)$ holds. This is the same global guard as that for expunge-route. That is not a coincidence, the two conditions should be the same, as shown by the first loop-formation scenario from Section 1.2. The actual AODVv2-04 protocol says that a node can re-join the protocol once `MAX_SEQNUM_LIFETIME` seconds have elapsed.

remove-link(H,K) Mark any routes through K as being broken, and send RERR messages accordingly

```
   true ==>
      for all nodes w:
        if (H.route[w].n = K) then
           H.route[w].e.x := Broken; // mark route as broken.
           multicast(RERR(m))    // propagate RERR to all neighbors
        endif
```

add-link(H,K) new link from H to K established. Do nothing.

4 Related Work and Conclusions

There is a long history of research on inductive and compositional analysis applied to network protocols: the work in [13,15,3,16,4] is representative. The contribution of this work is to apply these ideas to the verification of a protocol operating under dynamic, adversarial network changes. Our proof technique is standard (cf.[4]): we postulate an assertion and show that it is inductive by proving that it is preserved by every action. However, there are interesting aspects to the structure of the proof. Most importantly, our proof technique is 'local', that is, it is applied to a generic protocol node (or edge), and considers interference from only the nodes in the neighborhood of that node (or edge) during protocol execution. Hence, the method is compositional. It relies on symmetry in the sense that the generic node analyzed represents any of the nodes that may arise during the execution of the actual protocol. In addition, the possibility of adversarial network change is taken care of by modeling the changes as non-deterministic

actions, which are always enabled, and may take effect at any point. In [12] a simpler model of AODVv2 was analyzed. In particular, the AODVv2 model in that earlier work did not incorporate node restarts or the expunging of `Expired` route table entries. This meant that the earlier model did not need to consider the AllClear global guard. We note that consideration of `Expired` routes leads to the first example of a routing loop in Section 1.2.

The formation of routing loops has been studied for earlier forms of the AODVv2 protocol (AODV and DYMO) in [1,5,17,11] and [18]. Although it operates in the same environment and has the same goals, the version of AODVv2 under development differs significantly, in part due to efforts made to ensure that routing loop scenarios discovered for earlier forms are avoided. For instance, the use of sequence numbers in AODVv2 is completely different from that in AODV and DYMO. The version of AODVv2 (DYMO) analyzed in [9] by model checking fixed configurations allows intermediate, non-target nodes to generate RREQs, this is not possible in AODVv2-04. Nonetheless, some key features have been retained across the protocol versions. An important one is the use of (sequence number, hopcount) as a metric to ensure loop freedom. That is to be expected, as the intuition given in all of the protocol descriptions is that the sequence number represents the "freshness" of a route, while hopcount represents its "cost". Our work shows that this intuition is valid; but it also shows (from the loop formation scenarios) that care must be taken when considering disruptive network changes. We have found it surprisingly easy to construct the proof, and we suspect that this is so because of a focus, through compositional reasoning, on 'local' state rather than 'global' state, and the many simplifications introduced by the designers.

The AODVv2 model verified here represents a possible abstract protocol implementation. However, several features or options of the full protocol are either not modeled or are not modeled in their full generality. For instance, in our version each addressable entity in the network is, if present, identified with a single node in any network topology. In contrast, in the full AODVv2 protocol, entities may be 'multi-homed', and therefore messages sent to the entity may be sent to multiple destinations.

Another significant difference is that in the model, we assume that the metric used by all nodes to determine the 'least cost route' to a destination is based on hop count. That is, the distance between any two neighboring nodes is 1, and the cost of a path from node O to node T is the number of nodes in the path minus 1. The protocol actually allows protocol implementers to choose a different metric, which changes the 'least cost route'. In practice, such metrics may include information relating to bandwidth of individual edges connecting neighboring nodes, or the implementation of individual edge connections (wireless, wired, etc.), to name just a few possible metrics.

In addition, we note that the full AODVv2 protocol allows great scope for implementation decisions in the following form. Many per-node protocol decisions are described as 'must' but some are described as 'may.' For instance, if the route from node H to node T is marked as 'expired' in the route table of

H then H *must* not advertise this route to its neighbors. However, if H receives an `RREQ` for T from a neighbor G then H *may* choose to add this route to O to H's routing table and advertise the `RREQ` message to H's neighbors. In our analysis, we model these decisions as *must* instructions. Hence, any `RREQ` message received at a node H will be processed at H and forwarded to H's neighbors. We note that, the models described in our work represent models allowed by the AODVv2 protocol and therefore any errors or discrepancies found in the modelled protocol would represent discrepancies in the full AODVv2 protocol.

There are several other approaches to the analysis of dynamic and ad-hoc networks. The work in [2] shows that Hoare triples for restricted logics are decidable. Work in [8,6] applies well-quasi-ordering (wqo) theory to ad-hoc networks, while the algorithm of [7] relies on symbolic forward exploration, as does (in a different way) the method of [17]. It would be interesting to see how well these algorithmic and semi-algorithmic methods apply to the AODVv2 model. Our own recent work [12] shows that the loose coupling forced by dynamic network changes contributes to the effectiveness of compositional reasoning and local symmetry reduction.

4.1 Conclusion and Future Work

We describe a formal proof of loop-freedom for a model of the AODVv2 protocol. In the course of doing so, we discovered a mistake in version 04 of the protocol, which has been acknowledged and corrected by the designers. The straightforward nature of the proof strengthens the conjecture which originally inspired this work: that dynamic network protocols must be loosely coupled and, hence, especially amenable to inductive compositional analysis.

There are several open questions that remain. For instance, we are interested in techniques for the automatic generation of induction compositional assertions for use in the analysis of loosely coupled dynamic systems. Other questions surround the analysis of AODVv2 itself. One is to check whether the chosen values for timing constants are correct for a given configuration of the protocol; the work in [10] can be a good starting point. Another is to generalize this proof to apply to a richer class of distance metrics, as well as to network features such as multi-homing. A particularly important question is to find a good strategy for constructing proofs for the various combinations of "may" options which are permitted by the protocol, while avoiding a combinatorial explosion of protocol variants. As nearly all network protocols include a number of may options, this is a broadly applicable question, and especially relevant in practice.

Acknowledgments: We would like to thank the authors of the AODVv2-04 protocol, in particular Charles Perkins, for helpful comments on the loop-formation scenarios and the proof.

References

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

2. A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 690–705. Springer, 2007.
3. K. Chandy and J. Misra. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4), 1981.
4. K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
5. S. Das and D. L. Dill. Counter-example based predicate discovery in predicate abstraction. In *FMCAD*, volume 2517 of *Lecture Notes in Computer Science*, pages 19–32, 2002.
6. G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
7. G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of safety properties in ad hoc network protocols. In *PACO*, volume 60 of *EPTCS*, pages 56–65, 2011.
8. G. Delzanno, A. Sangnier, and G. Zavattaro. Verification of ad hoc networks with node and communication failures. In *FMOODS/FORTE*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012.
9. S. Edenhofer and P. Höfner. Towards a rigorous analysis of AODVv2 (DYMO). In *20th IEEE International Conference on Network Protocols, ICNP 2012, Austin, TX, USA, October 30 - Nov. 2, 2012*, pages 1–6. IEEE, 2012.
10. A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. Automated analysis of AODV using UPPAAL. In C. Flanagan and B. König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 2012.
11. P. Höfner, R. J. van Glabbeek, W. L. Tan, M. Portmann, A. McIver, and A. Fehnker. A rigorous analysis of AODV and its variants. In *MSWiM*, pages 203–212. ACM, 2012.
12. K. S. Namjoshi and R. J. Trefler. Analysis of dynamic process networks. In *TACAS 2015*, volume 9035 of *LNCS*. Springer-Verlag, 2015.
13. S. S. Owicki and D. Gries. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM*, 19(5):279–285, 1976.
14. C. Perkins, S. Ratliff, and J. Dowdell. IETF MANET WG Internet Draft: `http://datatracker.ietf.org/doc/draft-ietf-manet-aodvv2`, current revision 06, December 2014.
15. A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.
16. A. Pnueli. In transition from global to modular reasoning about programs. In *Logics and Models of Concurrent Systems*, NATO ASI Series, 1985.
17. M. Saksena, O. Wibling, and B. Jonsson. Graph grammar modelling and verification of ad hoc routing protocols. In *Theory and Practice of Software: TACAS 08*, LNCS, pages 18–32, 2008.
18. R. J. van Glabbeek, P. Höfner, W. L. Tan, and M. Portmann. Sequence numbers do not guarantee loop freedom – AODV can yield routing loops –. In *MSWiM*, page 10pgs. ACM, 2013.