Preliminary Description of NACK-based Ad-hoc On-demand Distance Vector Routing Protocol for MANETs

Alessandro Bianchi, Sebastiano Pizzutilo and Gennaro Vessio
Department of Informatics, University of Bari, Bari, Italy

Keywords: MANET, AODV Variant, Abstract State Machines.

Abstract: The present paper proposes a variant of the Ad-hoc On-demand Distance Vector (AODV) routing protocol for Mobile Ad-hoc NETworks (MANETs) by means of an Abstract State Machine (ASM)-based model. The variant introduces a new unicast message, which makes each host aware about the network topology more quickly than in the original AODV.

1 INTRODUCTION

A Mobile Ad-hoc NETwork (MANET, for short) is a network designed for wireless communications among nomadic hosts (Agrawal and Zeng, 2003). It does not need any fixed infrastructure, and communication sessions between source and destination are established and maintained by the cooperation of the hosts in the network. Since each host can directly communicate only within the area established by its transmission range, communications with external areas need the contribution of intermediate hosts according to a specific routing protocol. So, each host can act both as message producer and consumer in a communication session and as router supporting communications among other hosts. Moreover, during their lifetime, hosts can enter or leave the network at will and continuously change their relative position. So, the twofold role played by hosts in the network, as well as the continuous change of the network topology due to movement, requires the definition of specific routing protocols for properly managing the lack of a fixed infrastructure.

In this paper, we advance the proposal of a slight variant of the the Ad-hoc On-demand Distance Vector (AODV) routing protocol (Perkins et al., 2003): the NACK-based AODV (N-AODV). N-AODV makes each host aware about the network topology more quickly than AODV. The new algorithm is formally specified by means of the Abstract State Machine (ASM) formalism (Gurevich, 2000), and its correctness is proved.

The rest of this paper is structured as follows: Section 2 is about related work; Section 3 provides a background knowledge on both AODV and ASMs; Section 4 deals with N-AODV and its ASM-based model; finally, Section 5 concludes this paper and depicts the future development of our research.

2 RELATED WORK

AODV is one of the most popular routing protocol for MANETs, and many variants have been proposed aimed at its improvement. In most cases, modifications deal with security and performance. For example, an improvement for ensuring protection against blackhole attacks is proposed in (Lakshmi et al., 2010); and an optimization aimed at reducing cost, delay and packet loss is presented in (Lanjewar and Gupta, 2013). In both cases, the improvement is proposed according to non-formal approaches.

The usefulness of the formal approach in this domain is emphasized in (Nakhaee et al., 2011), where a method for route selection in AODV is provided thanks to Coloured Petri nets; and in (Höfner et al., 2012), where an analysis of AODV and two variants is conducted through AWN (Algebra for Wireless Network), a process algebra specifically tailored for this scope. These two variants concern the non-optimal route selection and the failure in discovering routes.

More in general, formal validation and verification of MANETs' behavior guarantees reliable results. For example, in (Singh et al., 2010) a process calculus for reasoning about MANETs is proposed. Similar approaches are followed in (Delzanno et al., 2010), where processes are represented by finite state machines; and in (Bianchi and Pizzutilo, 2010) and

Bianchi A., Pizzutilo S. and Vessio G..
Preliminary Description of NACK-based Ad-hoc On-demand Distance Vector Routing Protocol for MANETs.
DOI: 10.5220/0005105305000505
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 500-505
ISBN: 978-989-758-036-9

(Bianchi and Pizzutilo, 2013), where authors use Petri nets-based models for investigating some properties.

Finally, to our best knowledge, the ASM-based approach has been used in the MANET domain only for specifying location services and position-based routings among known locations (Benczúr et al., 2003). However, we choose ASMs mainly because of three different reasons. When the model expressivity is considered, literature agrees that ASMs show versatility in capturing both sequential and parallel computations, e.g. (Németh, 2002). Secondly, considering methodological issues, the ASM formalism has been successfully applied in both academia and industry for the design and the analysis of complex systems in several domains, and a specific development method got prominence in the last years (Börger and Stärk, 2003). Thirdly, considering the implementation point of view, the capability of translating formal models into executable code is provided by several tools, for example CoreASM (Farahbod et al., 2007).

3 BACKGROUND

3.1 AODV Routing Protocol

AODV is a *reactive* protocol that discovers and maintains routes on-demand, i.e. routes are built only as desired by source nodes using a route request/route reply cycle, which allows updating routing tables stored in each node (Perkins et al., 2003). When an *initiator* host needs to start a communication session to a *destination* node, and it does not know a proper route, it broadcasts a route request (RREQ) packet to all its neighbors. An RREQ packet includes, among the others: *initiator address* and *broadcast id* (this pair uniquely identifies the packet); *destination address*; *destination sequence number*, which expresses the freshness of the information about destination; and *hop count*, which expresses the distance. Because of broadcast transmissions, each intermediate node can receive several instances of the same RREQ: for avoiding the packet duplication, yet received RREQs with the same identifying pair are discarded.

Knowledge of routes is stored into routing tables, recorded into a cache memory of each node. More precisely, a routing table in a node lists all other nodes in the network, and the best (known) route to reach each of them. To this end, each entry of the routing table includes the address of the node, its sequence number, the hop count to reach it, and the *next hop* field identifying the next node in the route to reach it.

When a node n receives an RREQ, it updates information about initiator and about the host that directly sent that RREQ to n. Then, n checks if one of the following holds: it is the destination, or the destination is one of its neighbors, or it knows a route to the destination with corresponding sequence number greater than or equal to the one contained in the RREQ (this means that its knowledge about the route is recent). If so, n unicasts a route reply (RREP) packet back to initiator; otherwise, it updates the hop count field, and rebroadcasts the RREQ.

The process is so reiterated until a route to destination is found, or until the route discovery process times out. An RREP packet contains: *initiator* and *destination address*; *destination sequence number*; and *hop count*. While RREP travels towards initiator, routes are set up inside the routing tables of the traversed hosts. Once initiator receives the RREP, communication simply starts.

The protocol also includes a mechanism for recording the up-to-date information about the broken physical links, but this issue is outside our purposes.

3.2 Abstract State Machines

Informally speaking, ASMs are finite sets of so-called *rules* of the form **if** *condition* **then** *updates* (possibly with the **else** clause) which transform *abstract* states (Börger and Stärk, 2003). The concept of abstract state extends the usual notion of *state* occurring in finite state machines: it is an arbitrary complex structure, i.e. a domain of objects with functions and relations defined on them. On the other hand, the concept of rule reflects the notion of *transition* occurring in traditional transition systems: *condition* is a first-order formula whose interpretation can be true or false; while *updates* is a finite set of assignments of the form $f(t_1,\ldots t_n) := t$, whose execution consists in changing in parallel the value of the specified functions to the indicated value. In each state, all conditions are checked, so that all updates in rules whose conditions evaluate to true are simultaneously executed, and the result is the transition of the machine from one state to another.

The framework also includes constructs aimed at directly supporting refinements to parallel or distributed implementations; among them, of particular interest for us, is the **forall** construct, which allows executing all rules satisfying a given condition. Also, there are constructs aimed at supporting the mechanism of procedure calls; this is achieved by the definition of ASM *submachines*, i.e. parameterized rules, which allow the declaration of *local* functions, so that each call of a submachine works with its own instantiation of its local functions.

A generalization of basic ASMs is represented

by Distributed ASMs (DASMs) (Börger and Stärk, 2003), capable to capture the formalization of multi-agent systems. Essentially, a DASM is intended as an arbitrary but finite number of independent agents, each executing its own underlying ASM. In a DASM, the keyword **self** is used for supporting the relation between local and global states and for denoting the specific agent which is executing a rule.

4 NACK-BASED AODV

4.1 Description

In the original formulation of AODV, when an intermediate node n receives an instance of an RREQ and does not know a proper route to the specific destination, it simply rebroadcasts the RREQ. A bit more refined mechanism is the NACK-based AODV (N-AODV): in addition to rebroadcasting the RREQ, n unicasts a NACK (Not ACKnowledgement) packet back to initiator. The NACK is so used to inform all nodes beetwen n and initiator that, roughly speaking, n "does not know anything" about the destination. Each NACK packet includes the IP addresses and the sequence numbers of n and of initiator.

In N-AODV each host updates its own routing table when it receives an RREQ or an RREP, according to AODV, and when it receives a NACK. In this case, N-AODV updates the entries concering all nodes sending the NACK. So, the usage of NACKs allows improving the network topology awareness of each host. Moreover, if in a finite amount of time initiator receives only NACKs, then it knows that destination is currently outside the MANET space; if initiator does not receive any packet (neither NACKs or RREPs), then it can realize it is isolated.

4.2 ASM-based Model

A MANET that adopts N-AODV can be modeled by a DASM including a set of $agents = \{a_1, \ldots, a_n\}$, where each agent models the behavior of each node. In general, a host is characterized by several features, e.g. the amplitude of its transmission range, the direction and the speed of its movement, and so on. However, for our purposes, these features can be abstracted away, and we only consider the IP address, which univocally identifies each a_i in the MANET.

Since all agents implement the same protocol, each agent behaves according to the same ASM, so only one ASM is discussed in the following. Each ASM can be in one of the following states:

idle: the agent is inactive. Its configuration, $\forall$ *dest* $\in$ *agents*, is given by: *wishToInitiate*(**self**, *dest*) = false; *receivedRREQ*(**self**, *dest*) = false; *isEmpty*(*replies*(**self**)) = true; and *isEmpty*(*nacks*(**self**)) = true;

initiator: the agent has to start a new communication session. It is characterized by *wishToInitiate*(**self**, *dest*) = true;

router: the agent has received an RREQ. It is characterized by *receivedRREQ*(**self**, *dest*) = true;

forwarding: the agent is forwarding an RREP or a NACK to another destination. It is characterized by *isEmpty*(*replies*(**self**)) = false or *isEmpty*(*nacks*(**self**)) = false.

It is worth noting that a *destination* state is not necessary: a host knows to be the destination when, in *router* state, it receives an RREQ directed to it.

The functions dealing with the *initiator* and *router* states are:

wishToInitiate: $agents \times agents \rightarrow boolean$, indicating whether a new communication session to *dest* is required by the environment;

receivedRREQ: $agents \times agents \rightarrow boolean$, indicating whether an RREQ packet has been received.

Moreover, for the sake of simplicity, each agent is associated with three types of queues of messages: *requests*, *replies*, and *nacks*, which include RREQ, RREP, and NACK packets, respectively. This allows us working with abstract messages, simply inserting or deleting them into the corresponding queue. Each RREQ, RREP, or record (i.e. an entry of the routing table) is built by concatenation of the information described in Section 3.1 (in the pseudo-code the dot notation helps in identifying the value of a specific field of a packet). The functions dealing with the queues are defined as follows:

requests: $agents \rightarrow \{requestFrom(n) \mid n \in neighb\}$, represents the queue storing the RREQs received from the other nodes;

replies: $agents \rightarrow \{replyFrom(n) \mid n \in agents\}$, represents the queue storing the RREPs received from the other nodes;

nacks: $agents \rightarrow \{nackFrom(n) \mid n \in agents\}$, represents the queue storing the NACKs received from the other nodes;

isEmpty: $\{requests, replies, nacks\} \rightarrow boolean$, evaluates to true if the corresponding queue is empty; false otherwise;

top: $\{requests, replies, nacks\} \rightarrow \{requestFrom(n), replyFrom(n), nackFrom(n)\}$, is used for referring the first element of the corresponding queue.

Note that when the MANET starts operating, each agent is idle, i.e. for each agent both *wishToInitiate*(**self**, *dest*) and *receivedRREQ*(**self** *dest*) evaluate to false for each *dest*; and both *isEmpty*(*replies*(**self**)) and *isEmpty*(*nacks*(**self**)) evaluate to true.

In addition, each ASM includes the following:

neighb: *agents* → PowerSet(*agents*), specifies the nodes in the neighborhood of each agent;

routingTable: *agents* → PowerSet(*records*), represents the information about the nodes recorded into the agent's routing table. Whenever a node checks its routing table, it considers only the most recent information.

The values of the *neighb* and *routingTable* functions, as well as the set *agents*, depend on the particular scenario: they are dynamically set according to the MANET evolution, with respect to both the host mobility and the computational history.

The ASM pseudo-code of the *i*-th agent is:

```
AGENTPROGRAM(a_i) =
 if wishToInitiate(self, dest) = true then
  Initiator(dest)
 if ¬isEmpty(requests(self)) then {
  let dest = top(requests(self)).dest
  let nextHop = sender of top(requests(self))
  update routingTable(self)
  receivedRREQ(self, dest) := true
  Router(dest, nextHop)
 }
 if ¬isEmpty(replies(self)) {
  let dest = top(replies(self)).dest
  let nextHop = select c.nextHop
   from routingTable(self)
   with dest = c.dest
  update routingTable(self)
  UnicastRREP(nextHop)
  dequeue top(replies(self))
 }
 if ¬isEmpty(nacks(self)) {
  let dest = top(nacks(self)).dest
  let nextHop = select c.nextHop
   from routingTable(self)
   with dest = c.dest
  update routingTable(self)
  UnicastNACK(nextHop)
  dequeue top(nacks(self))
 }

UnicastRREP(n) =
 enqueue replyFrom(self) into replies(n)

UnicastNACK(n) =
 enqueue nackFrom(self) into nacks(n)
```

In the pseudo-code above, the instruction "update *routingTable*(**self**)" is not specified: it simply indicates that the agent's routing table is updated according to the received packet, RREQ or RREP or NACK.

Informally speaking, each agent is inactive until a new communication session is required by the environment; or until its computation is solicited by the receipt of an RREQ; or until it has to forward a unicast packet (either RREP or NACK) to the next hop in the route to reach the receiver of that packet. Activation of an agent unfolds different computational branches: two of them lead to the execution of a new instance of the *Initiator* or *Router* submachine, respectively; in the other two cases, forwarding activities of RREPs and NACKs, respectively, are executed. It is worth noting that all these activities evolve concurrently.

4.2.1 Initiator

The ASM pseudo-code of the *Initiator* submachine is:

```
Initiator(dest) =
 if dest ∉ neighb(self) ∧ dest ∉ routingTable(self)
 then {
  BroadcastRREQ
  sentRREQ(self) := true
 }
 else {
  StartCommunicationSession(dest)
  stopInitiator
 }
 if sentRREQ(self) ∧ ¬isEmpty(replies(self))
 then {
  select r from replies(self)
   with maximum destSeqNum
  StartCommunicationSession(dest)
  empty replies(self)
  stopInitiator
 }
 if sentRREQ(self) ∧ isEmpty(replies(self)) ∧
 ¬(timeout(self) = 0) then
  timeout(self) := timeout(self)−1
 if sentRREQ(self) ∧ ¬isEmpty(nacks(self))
 then {
  update routingTable(self)
  empty nacks(self)
 }
 if sentRREQ(self) ∧ timeout(self) = 0 then {
  stopInitiator
 }

BroadcastRREQ =
 forall n ∈ neighb(self) do {
  forall r ∈ requests(n) do {
   if requestFrom(self).dest = r.dest ∧
```

```
    requestFrom(self).id = r.id then
      discard requestFrom(self)
   }
   enqueue requestFrom(self) into requests(n)
  }

 stopInitiator =
  wishToInitiate(self, dest) := false
  sentRREQ(self) := false
```

In the pseudo-code above *StartCommunicationSession*(*dest*) is not described because it is not strictly part of the protocol.

The *Initiator* submachine is characterized by four local functions: *sentRREQ*: *agents* $\rightarrow$ *boolean*, acting as a flag indicating whether an RREQ has been sent; *timeout*: *agents* $\rightarrow$ *integer*, which models the maximum waiting time for RREPs; and the aforementioned *replies* and *nacks* queues. A new queue of RREPs and NACKs is instantiated for each specific communication session. This submachine includes two additional states:

waiting: it indicates that the agent is waiting for responses concerning that *dest* from the other agents in the DASM. Its configuration is given by: *wishToInitiate*(**self**, *dest*) = true; *sentRREQ*(**self**) = true; *timeout*(**self**) > 0;

endInitiating: it indicates that the computational activities executed by initiator, concerning the route discovery for that *dest*, are completed. Its configuration is: *wishToInitiate*(**self**, *dest*) = false; *sentRREQ*(**self**) = false.

If a route to *dest* is known, then the communication session simply starts; otherwise, *BroadcastRREQ* is executed. Its result consists in inserting a new RREQ into the *requests* queue of all the agent's neighbors and in evolving the current state to *waiting*. When an RREP is received (i.e. *isEmpty*(*replies*(**self**)) evaluates to false), then the computation continues; otherwise, when the timeout expires, the route discovery process ends. If a NACK is received, the routing table is updated. The *stopInitiator* rule simply resets the configuration to the *endInitiating* state.

4.2.2 Router

The ASM pseudo-code of the *Router* submachine is:

```
 Router(dest, nextHop) =
  if dest = self ∨ dest ∈ neighb(self) ∨
  dest ∈ routingTable(self) then {
   UnicastRREP(nextHop)
   dequeue top(requests(self))
   receivedRREQ(self, dest) := false
  }
  else {
   BroadcastRREQ
   UnicastNACK(nextHop)
   dequeue top(requests(self))
   receivedRREQ(self, dest) := false
  }
```

Note that *BroadcastRREQ* behaves as well as in the *Initiator* submachine; instead, *UnicastRREP* and *UnicastNACK* behave as well as in the main program.

The *Router* submachine includes the state *endRouting*, which specifies that the execution of the routing activities due to the route discovery process is completed. This state is only characterized by the value false for the *receivedRREQ*(**self**, *dest*) function.

If router is the destination of the RREQ (*dest* evaluates to **self**) or if it knows a route to dest ($dest \in neighb(\textbf{self}) \vee dest \in routingTable(\textbf{self})$), then it unicasts an RREP packet back to initiator. Otherwise, it rebroadcasts the RREQ to all its neighbors and unicasts a NACK packet back to initiator. In both cases, the computation evolves to the *endRouting* state for that value of *dest*.

4.3 Correctness

The proof of the correctness of N-AODV is quite simple, so, for the sake of brevity, it is only sketched.

First of all, it is worth noting that the execution of each ASM does not starve: both the program of each agent a_i, and *Initiator* and *Router* submachines allow their execution always evolve: even when they must wait for external events, the waiting time is finite. For what concerns the overall machine, both branches dealing with the unicast forwarding of packets restore the state, so that the computation can continue after their activities. The *Router* submachine surely returns back an RREP or a NACK packet, depending on the values of the guard conditions, and then evolves to the *endRouting* state. The *Initiator* submachine starts the communication session or enables neighbors to execute the protocol. If the latter happens, then, if initiator receives back an RREP, it appropriately continues the execution by starting the communication session; otherwise, it stops waiting for the timeout expiration. Depending on the receiving of NACKs, initiator possibly manages them.

Secondly, the pseudo-code shows that the correct packet is received back by initiator. In fact:

a. initiator receives back one or more RREPs if and only if a route to destination exists;

b. initiator receives back only NACKs but no RREPs if and only if no route to destination exists;

c. initiator does not receive back any packet if and only if it is isolated.

In order to prove issue (a) above, note that receiving RREPs means that the *replies* queue in the *Initiator* submachine is not empty, and this occurs if and only if the *UnicastRREP* rule has been executed, because only this rule enqueues an RREP into *replies*. In turn, the *UnicastRREP* rule is executed if and only if that router or one of its neighbors is the destination, or a route to destination is recorded into its routing table. Analogous proofs for issues (b) and (c).

Finally, it is easy to show that the execution of the protocol goes only through the described states: the execution of the rules transform the values of the functions only to the desired states. So, no unexpected behavior can occur.

5 CONCLUSION

In this paper, we have formally presented N-AODV: a variant of the AODV routing protocol for MANETs. The main advantage of the variant is that the hosts in the MANET can indirectly obtain information about the network topology more quickly than in AODV thanks to NACKs. In fact, in AODV the routing tables are updated on RREQ or RREP receiving, and more precisely, when the node n_i receives a packet sent by n_0 that traversed the chain $(n_0, n_1, \ldots, n_{i-1}, n_i)$, n_i updates its routing table only in entries concerning n_0 and n_{i-1}. Instead, N-AODV updates all entries $n_0, n_1, \ldots, n_i - 1$.

The research will continue with the aim to investigate how properly managing the knowledge carried on by NACKs and recording it into routing tables. Secondly, we are interested into formally analyze some computationally interesting properties, such as *starvation-freedom*, *reversibility* of the execution, and so on. Finally, if protocol performance issues are considered, it is worth noting that the use of NACKs injects overhead in the computation activities carried out by hosts, so, this overhead will be investigated through simulations, for evaluating if it is adequately balanced by the information gain obtained.

ACKNOWLEDGEMENT

This work has been partially funded by the Italian Ministry of Education, University and Research (MIUR), within the Piano Operativo Nazionale – PON02_00563_3489339.

REFERENCES

Agrawal, D. and Zeng, Q. (2003). *Introduction to Wireless and Mobile Systems*. Thomson Brooks/Cole.

Benczúr, A., Glässer, U., and Lukovszki, T. (2003). Formal Description of a Distributed Location Service for Mobile Ad Hoc Networks. In Börger, E., Gargantini, A., and Riccobene, E., editors, *Abstract State Machines*, volume 2589, pages 204–217. Springer.

Bianchi, A. and Pizzutilo, S. (2010). Studying MANET through a Petri Net-Based Model. In *2th International Conference of Evolving Internet*, pages 220–225.

Bianchi, A. and Pizzutilo, S. (2013). A Coloured Nested Petri Nets Model for Discussing MANET Properties. *International Journal of Multimedia Technology*, 3(2):38–44.

Börger, E. and Stärk, R. (2003). *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag.

Delzanno, G., Sangnier, A., and Zavattaro, G. (2010). Parameterized Verification of Ad Hoc Networks. In *21th International Conference of Concurrency Theory*, pages 313–327.

Farahbod, R., Gervasi, V., and Glässer, U. (2007). CoreASM: An Extensible ASM Execution Engine. *Fundamenta Informaticae*, 77(1–2):71–103.

Gurevich, Y. (2000). Sequential Abstract State Machines Capture Sequential Algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111.

Höfner, P., van Glabbeek, R., Tan, W., Portmann, M., McIver, A., and Fehnker, A. (2012). A Rigorous Analysis of AODV and its Variants. In *15th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 203–212.

Lakshmi, K., Priya, S. M., Jeevarathinam, A., Rama, K., and Thilagam, K. (2010). Modified AODV Protocol against Blackhole Attacks in MANET. *International Journal of Engineering and Technology*, 6(2):444–449.

Lanjewar, A. and Gupta, N. (2013). Optimizing Cost, Delay, Packet Loss and Network Load in AODV Routing Protocol. *International Journal of Computer Science and Information Security*, 4(11).

Nakhaee, A., Harounabadi, A., and Mirabedini, J. (2011). A Novel Communication Model to Improve AODV Protocol Routing Reliability. In *5th International Conference on Application of Information and Communication Technologies*, pages 1–7.

Németh, Z. (2002). Definition of a Parallel Execution Model with Abstract State Machines. *Acta Cybernetica*, 15(3):417–455.

Perkins, C., Belding-Royer, E., and Das, S. (2003). Ad-hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, http://tools.ietf.org/html/rfc3561.

Singh, A., Ramakrishnan, C., and Smolka, S. (2010). A Process Calculus for Mobile Ad Hoc Networks. In *10th International Conference on Coordination Models and Languages*, pages 440–469.