Intercepting Blackhole Attacks in MANETs: An ASM-based Model

Alessandro Bianchi, Sebastiano Pizzutilo and Gennaro Vessio

Department of Informatics, University of Bari, Via Orabona 4, 70125 Bari, Italy
{alessandro.bianchi, sebastiano.pizzutilo, gennaro.vessio}@uniba.it

Abstract. The inherent features of Mobile Ad-hoc NETworks (MANETs) make them vulnerable to various kinds of security attacks. In particular, in a so-called blackhole attack, one or more malicious hosts can send fake routing information towards an initiator, compromising the reliability of the network in the whole. In this paper, we propose a refinement of the NACK-based Ad-hoc On-demand Distance Vector (N-AODV) protocol, namely Blackhole-free N-AODV (BN-AODV), as a solution to intercept (cooperative) blackhole attacks in MANETs. Thanks to a formalization through an Abstract State Machine-based model, the correctness of the proposed protocol is formally proved.

Keywords: Abstract State Machines, Mobile Ad-hoc NETworks, Blackhole

1 Introduction

Mobile Ad-hoc NETworks (MANETs for short) are collections of nomadic hosts which communicate in a wireless way without the need of any fixed physical infrastructure [1]. The lack of infrastructure and the continuous topology change due to movement require the definition of specific routing protocols: in most cases a communication session between two hosts is set and maintained by a number of hosts lying in the path between the two communicants. All routing protocols for MANETs, in their basic definition, assume the trustworthiness of each host. However, because of its inherent features, in the presence of malicious hosts the reliability of a MANET is vulnerable to various kinds of attacks: flooding, wormhole, blackhole, and so on [2].

In a blackhole attack the malicious host sends fake routing information towards an initiator, claiming to know the best route to reach destination [3]. Packets are so routed towards the malicious host allowing it to misuse or discard them. Sometimes, the attack involves several malicious hosts that work cooperatively. Blackholes negatively impact the performance of a MANET, making worse the inherent problem of packet loss, due to mobility, for which there is no guarantee, in general, that a destination is reachable. Unfortunately, since it takes advantage of the characteristics of the protocols' mechanism, such an attack is easy to be carried out and intercepting it is difficult.

The literature proposes several solutions to intercept blackhole attacks, both single and cooperative ([2], [3] and [4] are surveys on this topic). Nevertheless, all these

solutions are described in a conventional (often informal) way, which is not rich enough to enable rigorous analysis about the capability of the protocol to intercept such attacks. Conversely, the availability of a high-level, formal environment could provide a way to analyze solutions to blackhole attacks rigorously.

To this end, the present paper proposes Blackhole-free NACK-based Ad-hoc On-demand Distance Vector (BN-AODV), a variant of the recently proposed NACK-based Ad-hoc On-demand Distance Vector (N-AODV) routing protocol [5], [6]. BN-AODV is here formally specified using Abstract State Machines (ASMs) [7], and its ability to intercept blackhole attacks is proved. In the following, several reasons for choosing the ASM formalism are presented. Firstly, it represents a general model of computation which subsumes all other classic computational models [8]. Secondly, it provides a way to describe algorithmic issues in a simple abstract pseudo-code, which can be translated into a high level programming language source code in a quite simple manner [9]. Moreover, considering methodological issues, it has been successfully applied for the design and analysis of critical and complex systems in several domains, and a specific development method came to prominence in the last years [7]. Finally, considering the implementation point of view, the capability of translating formal specifications into executable code, in order to conduct simulations of the models, is provided by tools like AsmL [10] (which is not maintained anymore), CoreASM [11] and ASMETA [12].

The rest of this paper is organized as follows. Section 2 summarizes the related literature. Section 3 overviews both ASM and N-AODV. Section 4 informally introduces BN-AODV, that is formally modeled in Section 5. Section 6 proves the correctness of the protocol. Section 7 concludes the paper and sketches future research.

2 Related work

Traditionally, the literature discusses MANET protocols simulating the network behavior (e.g., [13], [14]). The simulation-based approach is very effective from the execution viewpoint, mainly for evaluating performance and comparing different solutions. However, it takes into account only a limited, predictable range of scenarios and it is not able to formally prove properties of interest. Conversely, formal methods are useful for reasoning about MANET behavior and provide more reliable results. In literature, some process calculi specifically tailored for MANETs have been proposed, for example: the ω-calculus [15], CMN (Calculus of Mobile Ad Hoc Networks) [16], and AWN (Algebra for Wireless Networks) [17]. They capture essential characteristics of nodes: from their mobility to the packet broadcasting and unicasting. However, they are not directly executable, so, conducting simulations is not possible. Moreover, process calculi are typically based on mathematical notions that developers could find unfamiliar. Various general purpose state-based models have also been used in the MANET domain, for example, finite state machines [18] and Petri nets [19], [20]. In particular, Petri nets have been employed to study several issues: modeling and verification of routing protocols [21], [22]; evaluation of their performance [23]; application to vehicular networks [24]. With respect to process calculi, state-based models provide a comfortable way for representing algorithmic issues, especially from

the graphical point of view. Moreover, they are typically equipped with tools, such as CPN Tools [25], that allow one to execute the models. Nevertheless, state-based models lack of expressiveness: basically, they provide only a single level of abstraction and cannot support refinements to executable code. Moreover, according to [26], when compared to ASMs in modelling a variety of distributed systems, PNs are considered "neither intuitively clear nor formally simple". In order to overcome these limitations, our research makes use of Abstract State Machines (ASMs) [7].

The capability of ASMs to subsume other classic computational models has been stated in several works, e.g. [8], [27], [28], and an ASM sequential thesis has been proved in [29]. It states that ASMs suffice to capture the behavior of wide classes of sequential systems at any desired level of abstraction. This thesis has then been extended to parallel [30] and concurrent [31], [32] computations.

For what specifically concerns blackhole attacks, the literature includes several proposals, surveyed in [2], [3], but for the purposes of this work we only consider the solutions based on the AODV protocol and its variants. We can classify these solutions into two main categories, according to the approach they implement: topology-based and table-based. Solutions in the first category take into account the neighbourhood of the nodes in the MANET [33], or the existence of more than one route between a source and a destination [34], [35]. The second category ([36], [37], [38]) includes solutions in which nodes store information about all received packets in a new table. Except of [37], and [38], all these proposals are useless in case of cooperative blackhole. Instead, the solution we propose is able to identify both single and cooperative attacks.

3 Background

In the following, the main concepts concerning Abstract State Machines and NACK-based Ad-hoc On-demand Distance Vector routing protocol for Mobile Ad-hoc NETworks are summarized. Detailed descriptions can be found in the related literature.

3.1 Abstract State Machines

Informally speaking, Abstract State Machines are finite sets of so-called *rules* of the form **if** *condition* **then** *updates* which transform the *abstract* states of the machine [29]. 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, a 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.

According to [7], pairs of function names together with values for their arguments are called *locations*: they abstract the notion of memory unit. The current configuration of locations together with their values determines the current state of the ASM. 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 a transition of the machine from one state to another, i.e. from a configuration of values in locations to another. Moreover, for the unambiguous determination of the next state, updates must be *consistent*, i.e. no pair of updates must refer to the same location. The formalism also supports the mechanism of procedure calls; this is achieved by the definition of ASM *submachines*, i.e. parameterized rules. They support the declaration of *local* functions, so that each call of a submachine works with its own instantiation of its local functions in addition to the functions of the supermachine calling it.

Distributed ASMs (DASMs) [7] represent a generalization of basic ASMs: they capture the formalization of multiple agents acting in a distributed environment. Essentially, a DASM is intended as an arbitrary but finite number of independent agents (which are elements of a set *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.

In order to properly manage the complexity of the modeled systems, the ASM formalism includes several constructs [7]. For our purposes, we consider: the *let* rule, in the form **let** $x = t$ **in** P, aimed at assigning the value of t to x, and then executing the rule P; the *forall* rule (**forall** x **with** φ **do** P), which executes P in parallel for each x satisfying φ; the *choose* rule (**choose** x **with** φ **do** P), which chooses an x satisfying φ, and then executes P; the *seq* rule (P **seq** Q) which sequentially executes P and then Q.

3.2 NACK-based Ad-hoc On-demand Distance Vector Routing Protocol

The interception mechanism of blackhole attacks here proposed is derived from NACK-based Ad-hoc On-demand Distance Vector (N-AODV) [5]: it is a variant of the popular Ad-hoc On-demand Distance Vector (AODV) [39]. Its aim is to improve the network topology awareness of the network nodes through the adoption of a specific control packet named NACK (Not ACKnowledgement) [6]. N-AODV is a reactive protocol in which routes are discovered on-demand, and stored into *routing tables* within each node. The routing table associated with each node lists all the discovered (still valid) routes towards other nodes in the network and information on them. In particular, an entry of the routing table of the host i concerning a node j includes: the *address* of j; the last known *sequence number* of j; the *hop count* field expressing the distance between i and j; and the *next hop* field identifying the next node in the route to reach j. The sequence number is a monotonically increasing value maintained by each node: it helps other nodes to express the freshness of the information about it.

When an initiator wants to start a communication, it first checks if the destination is in its neighborhood (so that it is directly reachable), or a route to it is currently stored in its routing table. If so, the protocol ends and the communication simply starts. Otherwise, initiator fires the route discovery mechanism, by broadcasting route request (RREQ) packets to all its neighbors. Among the others, an RREQ packet includes: *initiator address* and *broadcast id*; *destination address*; *destination sequence number*, which expresses the latest available information about destination; and *hop count*, initially set to 0, and increased by each intermediate node. Note that the pair *initiator*

address and *broadcast id* uniquely identifies the packet; in this way, duplications of RREQs that nodes have handled before can be ignored.

When an intermediate node *n* receives an RREQ, it updates the routing table entry for initiator, concerning both the sequence number of initiator and its next hop field; if an entry for initiator does not exist, it is created. Then the process is reiterated: *n* checks if the destination is a neighbor, or if it knows a route to destination with corresponding sequence number greater than or equal to the one contained into the RREQ (this means that its knowledge about the route is more recent). In both cases, *n* unicasts a route reply (RREP) packet back to initiator; an RREP contains: *initiator* and *destination address*; *destination sequence number* and *hop count*. Otherwise, *n*: updates the hop count field; rebroadcasts the RREQ to all its neighbors; and unicasts a NACK packet back to initiator. The NACK is so used to inform all nodes between *n* and initiator that, roughly speaking, *n* "does not know anything" about destination. Each NACK packet includes the addresses and the sequence numbers of *n* and initiator and their distance.

The route discovery successfully ends when a route to destination is found. While the RREP travels towards initiator, routes are set up inside the routing tables of the traversed hosts by creating an entry for destination when needed. Once initiator receives the RREP, the communication session can start. Conversely, the route discovery fails when: no RREQ reaches a node which is in the destination neighborhood; or no RREQ reaches a node whose routing table contains a route to destination; or a previously set timeout expires while initiator is waiting for RREPs. The first two cases depend on the non-reachability of destination; instead, the last case can be due to either the isolation of the destination, or too long distances, or changes in the topology during the packet transmission.

4 Blackhole-free N-AODV

The solution here proposed for intercepting blackholes relies on N-AODV, and it adopts two additional control packets, namely *Challenge* (CHL) and *Response* (RES) packets, aimed at ensuring that no blackhole is in the discovered route. To this end, asymmetric cryptography is followed: every node *j* is associated with a public key (denoted by K_j) and a private key (denoted by K_j^{-1}). Data packets exchanged during communication sessions, as well as CHL and RES packets, that are to be routed to a node *j*, must then be encrypted with K_j, and only *j* can decrypt them by using the corresponding K_j^{-1}. Conversely, RREQ and RREP packets are not encrypted. The public keys are spread over the network by the RREQ and NACK packets. More precisely, both RREP and NACK are enriched with a field storing the public key of the host producing the packet. In this way, each node receiving these packets knows the public key associated with the issuer; in other words: when a node *i* learns about the identity of a node *j*, then *i* also learns K_j.

The main idea behind BN-AODV is that every intermediate node lying in the discovered route to reach destination is *responsible* of the trustworthiness of the next hop of such route. To this end, if a node *n* receives an RREP directed to initiator from one of its neighbours *m*, then *n* must check the trustworthiness of the received *RREP*.

This is accomplished by sending a *CHL* packet to destination *d*, through *m*, encrypted with K_d, The *CHL* can simply consist of a nonce to be decremented by *d*. If the *CHL* safely reaches *d*, then *d* must reply with a *RES* packet, encrypted with K_n, to be unicasted towards *n*. If *n* receives back the proper *RES*, then it considers the *RREP* to be trusted and forwards the packet to the next hop in the reverse route to reach initiator. The process is then reiterated by every intermediate node in the route, until the *RREP* reaches initiator, which checks the trustworthiness of the *RREP* for the last time. If the last *RES* is received, the communication session can start. Conversely, if an intermediate node *n*, waiting for a *RES*, does not receive back the expected packet, it suspects the next hop *m*, in the route to reach *d*, to be a blackhole. As a consequence, *n* stops forwarding the *RREP*. If destination is not safely reachable and no alternative secure route is discovered until the timeout expiration, the route discovery fails.

In the case of cooperative attack, it is worth noting that one or more colluders could confirm the trustworthiness of the fake *RREP*. Nevertheless, the approach here proposed is able to intercept such an attack thanks to the last control executed by initiator. In fact, in this way, the protocol is able to judge secure or not the entire route, so intercepting not only single blackholes but also cooperative attacks. Note that, in this case, BN-AODV detects only one of the malicious nodes involved in the attack. More precisely, suppose we have a group of colluders $c_1, c_2, ..., c_n$ and a main blackhole *b*, lying consecutively in the route to reach destination *d* between a host *h* and *d*, i.e. $h, c_1, c_2, ..., c_n, b, ..., d$. The fake *RREP* generated by *b* then goes through the chain $c_n, ..., c_2, c_1$, before reaching *h*. Thanks to the protocol, *h* is able to stop the route discovery, however it can only suspect that c_1 is malicious, without any possibility to deduct something about the next nodes of the chain. Nevertheless, the other malicious nodes can be detected, one by one, in successive route discovery executions.

The proposed approach is quite draconian: if a node does not forward the correct *RES* it is suspected to be a blackhole. Unfortunately, the loss of correct *RES* packets (*CHL* packets) may be caused by node movement: in this case an honest node may be incorrectly considered a blackhole. In order to mitigate this effect, we introduce *trust levels* associated with each node. If a node *n* suspects one of its neighbours *b* to be a blackhole, then *n* adds *b* in a *trust table*. The *b*-th entry of the trust table of a node *n* expresses the trust level *n* has gained about *b* during past executions of the route discovery. Every time a node is suspected to be a blackhole, its trust level is decreased; every time it forwards the correct *RES* packet, its trust level is increased. In this way, a node is not excluded from routing activities at once, but only when its trust level reaches a lower limit. We assume a lower limit equals to $-\theta$; when a node *i* discovers the existence of a node *j*, *i* creates an entry concerning *j* in its trust table and initializes the trust level of *j* to 0. Every time the trust level associated with *j*, drops to $-\theta$, *i* marks *j* as malicious. As a consequence, all packets coming in from *j* are always discarded by *i*, moreover *i* does not send packets to *j* anymore.

Finally, note that the use of cryptography is crucial to prevent blackholes from manipulating messages. In fact, without using keys, a blackhole could send fake responses. In other words, malicious nodes are forced to behave properly, otherwise their possible attacks are immediately intercepted. It is worth noting that both trust levels and tables, and cryptographic issues are not considered in the present model.

5 ASM-based Model

A MANET adopting BN-AODV can be modeled by a DASM including a set of *Hosts* = $\{h_1, ..., h_n\}$, where each h_i models the behavior of a single node executing the protocol. We can think about each h_i as univocally identified by its address. For the purposes of the present paper, each host behaves either as an honest node, or as a blackhole, or as a colluder, so three ASMs are presented, for describing the three different cases.

5.1 Honest ASM

All honest hosts behave homogeneously in accordance with the protocol, therefore we here discuss only one ASM. For space reasons, only the main issues are described below, leaving at an abstract level the less important details. Each *honest* ASM includes the following functions:

- *neighb*: *Hosts* → PowerSet(*Hosts*), which specifies the neighborhood of each host;
- *wishToInitiate*: *Hosts* → *boolean*, which indicates whether a new communication session is required;
- *initiateTo*: *Hosts* × *Hosts* → *boolean*, which specifies the destination of the desired communication;
- *hasToVerify*: *Hosts* × *Hosts* × *Hosts* → *boolean*, which establishes if the received RREP has to be verified or not;
- *trusted: Hosts* × *Hosts* → *boolean*, which acts as a flag to indicate whether an RREP packet concerning a destination has been judged to be trusted or not.

The meaning of the arguments of the functions above is quite obvious, except of *hasToVerify*: its first argument indicates the node to which the verified RREP must be forwarded; the second is the host from which the (un)verified RREP comes; the third is the destination of the current route discovery.

In order to model broadcasting and unicasting of packets, every node is associated with five queues of messages: *requests*, *replies*, *challenges*, *responses* and *nacks*, which include RREQ, RREP, CHL, RES and NACK packets, respectively. These queues are managed by the functions *isEmpty* and *top*, and by the rules *enqueue* and *dequeue* whose purpose is in accordance with the respective names. The access of a field *f* of a packet *p* is denoted by the form *p.f*. Note that in the protocol specification each control packet does not include information about the router forwarding it; however, in the following, we provide an additional field (*p.sender*) in each control packet for modelling reasons. Moreover, each node is associated with a *routing table* plus a *trust table* representing the information the nodes store about other nodes.

At the initial state of the computation all tables and queues are empty; the neighbourhood of each host is pre-set, depending on the initial MANET topology; all hosts are inactive, i.e. *wishToInitiate* and *initiateTo* evaluate to *false* for each node and for each pair of nodes, respectively; *hasToVerify* evaluates to *false* for each triple of nodes; *trusted* evaluates to *undef* for each pair of nodes.

The ASM pseudo-code of the *i*-th host is shown below, as *HostProgram*:

```
HostProgram ≡
  if ¬isEmpty(requests(self)) then {
    let RREQ = top(requests(self)), previousHop = RREQ.sender in
      UpdateRoutingTable(self, RREQ)
      Router(RREQ, previousHop)
      dequeue RREQ from requests(self)
  }
  if wishToInitiate(self) = true then
    forall dest ∈ Hosts with dest ≠ self do
      if initiateTo(self, dest) = true then
        Initiator(dest)
  if ¬isEmpty(replies(self)) then
    let RREP = top(replies(self)), nextHop = RREP.sender in
      if RREP.init ≠ self then {
        choose entry ∈ routingTable(self) with entry.dest = RREP.init
        previousHop := entry.nextHop seq
        hasToVerify(previousHop, nextHop, RREP.dest):=true
      }
  forall previousHop ∈ neighb(self) do
    forall nextHop ∈ neighb(self) do
      forall dest ∈ Hosts do
        if hasToVerify(previousHop, nextHop, dest) then
          Verify(top(replies(self)), previousHop, nextHop, dest)
  if ¬isEmpty(challenges(self)) then {
    let CHL = top(challenges(self)) in
      if CHL.dest = self then {
        let previousHop = CHL.sender in
          create_RES seq
          enqueue RES into responses(previousHop)
          dequeue CHL from challenges(self)
      }
      if CHL.dest ≠ self then {
        choose entry ∈ routingTable(self) with entry.dest = CHL.dest
        nextHop := entry.nextHop seq
        enqueue CHL into challenges(nextHop)
        dequeue CHL from challenges(self)
      }
  }
  if ¬isEmpty(responses(self)) then {
    let RES = top(responses(self)) in
      if RES.dest ≠ self then {
        choose entry ∈ routingTable(self) with entry.dest = RES.dest
        previousHop := entry.nextHop seq
        enqueue RES into responses(previousHop)
        dequeue RES from responses(self)
      }
  }
  if ¬isEmpty(nacks(self)) then {
    let NACK = top(nacks(self)) in
      if NACK.dest ≠ self then {
        choose entry ∈ routingTable(self) with entry.dest = NACK.dest
        previousHop := entry.nextHop seq
        UpdateRoutingTable(self, NACK)
        enqueue NACK into nacks(previousHop)
        dequeue NACK from nacks(self)
      }
  }
```

HR1 HR2 HR3 HR4 HR5 HR6 HR7

Informally speaking, each host is inactive as long as no rule is applicable; the idleness is left when one of the events guarding the conditions of the seven rules of the machine happens. These events concern: an RREQ is received, then the rule HR1 fires, leading to the execution of the *Router* submachine; the need to start a new communication session is required, leading to the execution of the *Initiator* submachine (rule HR2); an RREP is received, so HR3 establishes the need to verify its trustworthiness; HR4 executes the *Verify* submachine when needed; a CHL is received (HR5) so the node must reply with an RES (if the CHL was directed to that node), or it must simply forward the CHL; an RES is received (HR6), so it has to be forwarded if it is not directed to that node; a NACK is received (HR7), so it has to be forwarded if it is not directed to that node.

The *create_RES* statement in HR5 has the effect to generate a new RES packet: it is not formally specified for abstracting from the specific representation of the packet. Analogous statements occur in the machines described in the following.

Note that the *Router* and the *BroadcastRREQ* submachines are analogous to the homonymous machines used for modelling the N-AODV routing protocol we proposed in [5], so, for space reasons, they are not formally modelled here: it is sufficient to state that they behave accordingly to the respective names. Analogous considerations for the *UpdateRoutingTable* and *UpdateTrustTable* submachines. Therefore, in the following we only focus on the *Initiator* and *Verify* submachines: their ASM description will then be useful for proving the correctness of the proposed protocol.

The *Verify* submachine includes two local functions:

- *verify_waiting*: *Hosts* × *Hosts* → *boolean*, which acts as a flag indicating whether the host is still waiting for the RES directed to it. Its initial value is *false*;
- *verify_timeout: Hosts* × *Hosts* → ℕ, which models the waiting time for the RES.

The ASM pseudo-code of *Verify* is shown below:

```
Verify(RREP, previousHop, nextHop, dest) ≡
  if ¬verify_waiting(self, dest) then {
    create_CHL seq
    enqueue CHL into challenges(nextHop)
    verify_waiting(self, dest) := true
    verify_timeout(self, dest) := default_value
  }
  if verify_waiting(self, dest) then {
    if ¬isEmpty(responses(self)) then
      if ReliableRREP(self, top(responses(self))) then {
        trusted(self, dest) := true
        verify_waiting(self, dest) := false
        dequeue top(responses(self)) from responses(self)
      }
    verify_timeout(self, dest) := verify_timeout(self, dest) − 1
  }
  if verify_waiting(self, dest)∧verify_timeout(self, dest) =0 then {
    trusted(self, dest) := false
    verify_waiting(self, dest) := false
  }
  if trusted(self, dest) then {
    UpdateRoutingTable(self, RREP)
    UpdateTrustTable(self, nextHop)
    dequeue RREP from replies(self)
```

VR1 VR2 VR3 VR4

```
    if previousHop ≠ null then
      enqueue RREP into replies(previousHop)
  }
  if ¬trusted(self, dest) then {
    UpdateTrustTable(self, nextHop)
    dequeue RREP from replies(self)
  }
```

VR5

ReliableRREP (**self**, top(*responses*(**self**))) is a predicate that evaluates to *true* when the received RES packet is correct, so the respective RREP can be considered reliable. It simply verifies that the content of the RES packet is exactly the expected value.

The submachine includes five rules, aimed at: creating and sending the CHL (VR1); establishing (VR2) or denying (VR3) the trustworthiness of the previously received RREP; updating both the routing table and the trust table if the RREP is trusted (VR4); updating only the trust table if the RREP is not trusted (VR5). The *verify_waiting* function avoids the node to create the same CHL more than one time. While the node is waiting, it checks the incoming RESs: if an RES directed to it is received, and its correctness is verified, then the RREP is judged to be trusted. Otherwise, the timeout is decreased: when the timeout expires, the node assumes the *RREP* to be not trusted. Finally, note that VR4 (according to [7]) uses the special constant *null*, for indicating that *previousHop* ∉ *Hosts*: this happens when *Verify* is called by the *Initiator* submachine because the *RREP* must not be forwarded to any other host.

Similarly to *Verify*, the *Initiator* submachine includes the following local functions:

- *initiator_waiting*: *Hosts* × *Hosts* → *boolean*, which acts as a flag indicating whether initiator is still waiting for (at least) an RREP directed to it. Its initial value is *false*;
- *initiator_timeout*: *Hosts* × *Hosts* → ℕ, which models the waiting time for RREPs.

The ASM pseudo-code of *Initiator* is shown below:

```
Initiator(dest) ≡
  if dest ∈ neighb(self) ∨ dest ∈ routingTable(self) then {
    CommunicationSession(dest)
    initiateTo(self, dest) := false
  }
  if dest ∉ neighb(self) ∧ dest ∉ routingTable(self) then {
    create_RREQ seq
    BroadcastRREQ(RREQ)
    initiator_waiting(self, dest) := true
    initiator_timeout(self, dest) := default_value
  }
  if initiator_waiting(self, dest) then
    initiator_timeout(self,dest):=initiator_timeout(self,dest)−1 seq
    if ¬isEmpty(replies(self)) then
      forall r∈replies(self) with r.init=self and r.dest =dest do
        if trusted(self, dest) = undef then
          let nextHop = r.sender in Verify(r, null, nextHop, dest)
  if trusted(self, dest) then {
    CommunicationSession(dest)
    initiateTo(self, dest) := false
    initiator_waiting(self, dest) := false
  }
  if initiator_waiting(self, dest) ∧ ¬isEmpty(nacks(self)) then {
    forall n ∈ nacks(self) with n.dest = self do {
      UpdateRoutingTable(self, n)
```

IR1 IR2 IR3 IR4 IR5

```
    dequeue n from nacks(self)
  }
}
if initiator_waiting(self, dest) ∧ initiator_timeout(self, dest)=0
then {
  initiateTo(self, dest) := false
  initiator_waiting(self, dest) := false
}
```
IR6

If a route to destination is already known, the communication session simply starts (IR1). Otherwise, the route discovery process is initiated (IR2). Until the timeout is greater than 0 (IR3), the incoming RREPs are checked. If (at least) an RREP directed to the host is received, the last CHL is sent. If the last RES is received, then the communication session can start (IR4). If NACKs directed to the host are received during the waiting for RREPs (IR5), the routing table is updated. The last rule (IR6) simply resets the node to inactivity.

The initiator is the last host verifying the trustworthiness of the *RREP*, so in IR3 the *Verify* submachine is called with a *null* value, instead of *previousHop* argument.

Note that an initiator could receive multiple RREPs concerning the same *dest*; in order to reduce redundancy, the protocol requires to discard them once the reliability of a route has been verified. However, for space reasons, this computation is not described.

5.2 Malicious ASMs

The literature does not provide a univocal, formal definition of blackhole (and colluder). In the following, we firstly provide a precise definition for this concept; then we present a model of malicious ASM.

Definition 1 (**Forward neighbor**). A *forward neighbor* f_n of a node n is the next hop of n in the route to reach the destination d, i.e. $n, f_n, \ldots, d$.

Thanks to the notion of forward neighbor, a blackhole is recursively defined:

Definition 2 (**Blackhole**). A *blackhole* is: (*i*) a main blackhole if it originates fake RREPs; or (*ii*) a colluder if its forward neighbor is a blackhole.

Therefore, differently from honest hosts, malicious hosts can behave heterogeneously in accordance with two ASMs, depending on whether the host is the main blackhole or one of its colluders.

Note that for both the blackhole and colluder nodes we assume the worst possible scenario: they behave maliciously without executing any other route discovery activity. Moreover, we assume that malicious nodes never behave honestly. Finally, we assume that the respective queues are initially empty for both malicious ASMs.

Blackhole program. The *malicious* ASM of the host behaving as main blackhole acts as a router claiming to know the best route to reach destination. This is done independently from the knowledge the blackhole has about destination. Its model is:

```
BlackholeProgram ≡
  if ¬isEmpty(requests(self)) then {
    let RREQ = top(requests(self)), previousHop = RREQ.sender in
      UpdateRoutingTable(self, RREQ)
      MaliciousRouter(RREQ, previousHop)
      dequeue RREQ from requests(self)
```

```
  }
```

where the *MaliciousRouter* submachine is simply:

```
MaliciousRouter(RREQ, previousHop) ≡
  create_RREP seq
  enqueue RREP into replies(previousHop)
```

In words: every time an RREQ reaches the blackhole, its routing table is updated and a fake RREP is sent back: it is created in accordance with the information stored in that RREQ.

Colluder program. The colluder node a priori forwards RREP packets independently from their reliability. Its model is as follows:

```
ColluderProgram ≡
  if ¬isEmpty(replies(self)) then {
    let RREP = top(replies(self)) in
      if RREP.init ≠ self then {
        choose entry ∈ routingTable(self) with entry.dest = RREP.init
        let previousHop := entry.nextHop seq
          enqueue RREP into replies(previousHop)
        dequeue RREP from replies(self)
      }
  }
```

6 Correctness

The ASMs modelling the malicious nodes are quite simple, so it is not necessary to prove their correctness. Instead, this section aims at proving that possible attacks are intercepted in every protocol execution. To this end, we assume *perfect encryption*: encrypted messages cannot be read without knowing the corresponding decryption key; this assumption leads to state that the *Verify* submachine consider reliable only the RREPs actually produced by the proper hosts. Secondly, it is worth remarking that multiple route discoveries can be executed in parallel and that each host can participate in different route discoveries playing different roles. Moreover, we only consider single communication attempts between couples of nodes: this does not compromise generality. We firstly provide the following definition:

Definition 3 (**Backward neighbor**). A *backward neighbor* b_n of a node n is the next hop of n in the route to reach initiator i, i.e. $i, ..., b_n, n$.

The correctness of the protocol modelled by the ASMs in Sect. 5 is stated by the two following theorems: the first one assume that the network only includes one malicious host (i.e., no colluders are considered); the second theorem generalizes the claim to networks including colluders.

Theorem 1. *The honest hosts intercept any single blackhole attack.*

Proof. Let N be the set of network nodes, and let *fRREP* be the fake RREP produced by the blackhole $b \in N$. In order to prove the claim, it must be proved that *fRREP* is discarded by the backward neighbor, $n_k \in N$, of b. Let $n_0, n_1, ..., n_{k-1}, n_k, b$ the route between the initiator n_0 and the blackhole. Formally, it must be proved that *fRREP* $\notin$ *replies* (n_{k-1}). The only rule allowing n_k to enqueue an RREP into the *replies* queue of n_{k-1} is the rule VR4 in the *Verify* submachine. More precisely, the RREP is forwarded only if *trusted*(**self**, *dest*) evaluates to *true*; in turn, the value of this location is initially set to *undef*, and it is set to *true* only inside the VR2 of the same submachine, when the

proper RES is received back by n_k. Since *Verify* does not state the trustworthiness of *fRREP*, *trusted*(**self**, *dest*) evaluates to *false*, so the received RREP is discarded in the last line of VR4 (**dequeue** *RREP* **from** *replies*(**self**)). Thus, the protocol always intercepts single blackhole attacks. □

Theorem 2. *The honest hosts intercept any cooperative blackhole attack.*

Proof. Let k be the number of colluders $c_i \in N$, $(1 \leq i \leq k)$ of the main blackhole b. Each c_i forwards RREP packets without checking their trustworthiness, in accordance with *ColluderProgram*. In order to prove the claim, let's consider the worst scenario, in which all nodes between the initiator n_0 and b are colluders, i.e. the route is $n_0, c_1, c_2, \ldots, c_k, b$. It must be proved that when $fRREP \in replies(n_0)$ the communication does not start. If $fRREP \in replies(n_0)$ then n_0 executes the IR3 rule of the *Initiator* submachine. In turn, this rule executes the verification of the RREP, and the communication session can start only if that RREP is trusted, otherwise it is discarded: since *fRREP* is not trusted by definition, it is discarded, and the communication cannot start. The case in which there is at least one honest node in the route between n_0 and b is captured by Theorem 1. Therefore, the protocol always intercepts cooperative attacks. □

7 Conclusion

The diffusion of MANETs imposes the need to properly manage security issues. The present paper proposed the Blackhole-free NACK-based AODV (BN-AODV) routing protocol for intercepting blackhole attacks: differently from analogous proposals, we formally specified the protocol using Abstract States Machines and thanks to the formalism the capability to identify *all* nodes responsible of the attack has been proved.

Unfortunately, the formal static approach followed in this paper is not sufficient for evaluating the capability of the protocol to identify *only* malicious nodes. In fact, the draconian approach to the discovery of blackholes could label an honest node as malicious, simply because of the change of the topology, due to the hosts' movements.

The next step of the research will implement the model into the ASMETA tool [12] with a twofold purpose. On one hand, the execution of the model in a proper environment will allow us to discover possible errors and to identify improvements: for example, thanks to the implementation, the concept of time (e.g., the time nodes must wait for the proper responses) will be better treated in order to consider real time events. On the other hand, the behaviour of the protocol can be empirically investigated.

References

1. Agrawal, D., Zeng, Q.: Introduction to Wireless and Mobile Systems. Thomson Brooks/Cole (2003)
2. Kannhavong, B., Nakayama, H., Nemoto, Y., Kato, N., Jamalipour, A.: A Survey of Routing Attacks in MANET. IEEE Wireless Comm., 14(5), 85–91 (2007)
3. Tseng, F.H., Chou, L.D., Chao, H.C.: A survey of black hole attacks in wireless MANET. Human-centric Computing and Information Sciences, 1:4 (2011)

4. Agrawal, P., Ghosh, R.K., Das, S.K.: Cooperative Black and Gray Hole Attacks in Mobile Ad Hoc Networks. In: 2nd International Conference on Ubiquitous Information Management and Communication, 310–314 (2008)
5. Bianchi, A., Pizzutilo, S., Vessio, G.: Preliminary Description of NACK-based Ad-hoc On-demand Distance Vector Routing Protocol for MANETs. In: 9th Int. Conf. on Software: Engineering and Applications, pp.500-505. (2014)
6. Bianchi, A., Pizzutilo, S., Vessio, G.: CoreASM-based Evaluation of the N-AODV Protocol for Mobile Ad-hoc NETworks. J. of Mobile Multimedia, 12, 31-51 (2016)
7. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag (2003)
8. Gurevich, Y.: Evolving Algebras 1993: Lipari Guide. Börger, E., ed., Specification and Validation Methods, Oxford University Press, pp. 9-36 (1995)
9. Börger, E.: High-level system design and analysis using Abstract State Machines. Hutter, D., Stephan, W., Traverso, P., Ullmann, M., eds., Current Trends in Applied Formal Methods, LNCS 1641, Springer-Verlag, pp. 1-43 (1999)
10. Gurevich, Y., Rossman, B., Schulte, W.: Semantic Essence of AsmL. Theoretical Computer Science, 342(3), pp. 370-412 (2005)
11. Farahbod, R., Gervasi, V., Glässer, U.: CoreASM: An Extensible ASM Execution Engine. Fundamenta Informaticae, 77(1-2), pp. 71-103 (2007)
12. Gargantini, A., Riccobene, E., Scandurra, P.: Model-Driven Language Engineering: The ASMETA Case Study. Proc. of the 3rd International Conference on Software Engineering Advances, pp. 373-378 (2008)
13. Jayakumar, G., Gopinath, G.: Performance Comparison of Two On-demand Routing Protocols for Ad-hoc Networks Based on Random Way Point Mobility Model. American Journal of Applied Sciences, 5(6), 659–664 (2008)
14. Goyal, P.: Simulation Study of Comparative Performance of AODV, OLSR, FSR and LAR Routing Protocols in MANET in Large Scale Scenarios. In: World Congress of Information and Communication Technologies 283–286 (2012)
15. Singh, A., Ramakrishnan, C., Smolka, S.: A Process Calculus for Mobile Ad Hoc Networks. In: 10th Int.Conf. Coordination Models and Languages, 296–314 (2008)
16. Merro, M.: An Observational Theory for Mobile Ad Hoc Networks. Information and Computation, 207(2), 194–208 (2009)
17. Fehnker, A., Glabbeek, R.V., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: A Process Algebra for Wireless Mesh Networks. In: 21st European Symposium on Programming, 295–315 (2012)
18. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized Verification of Ad Hoc Networks. In: 21th Int. Conference of Concurrency Theory, 313–327 (2010)
19. Bianchi, A., Pizzutilo, S.: Studying MANET through a Petri Net-Based Model. In: 2th International Conference of Evolving Internet, 220–225 (2010)
20. Bianchi, A., Pizzutilo, S.: A Coloured Nested Petri Nets Model for Discussing MANET Properties. Int. Journal on Multimedia Technology, 3(2) 38-44 (2013)
21. Xiong, C., Murata, T., Tsai, J.: Modeling and simulation of routing protocol for MANET using Colored Petri Nets. In: Conf. Application and Theory of Petri Nets: Formal Methods in Soft. Eng. and Defence Systems, Vol. 12, 145–153 (2002)

22. Xiong, C., Murata, T., Leigh, J.: An approach for verifying routing protocols in mobile ad hoc networks using Petri nets. In: 6th IEEE Symposium on Circuits and Systems, Vol. 2, 537–540 (2004)
23. Erbas, F., Kyamakya, K., Jobmann, K.: Modelling and performance analysis of a novel position-based reliable unicast and multicast routing method using coloured Petri nets. In: Vehicular Technology Conf., Vol. 5, 3099–3104 (2003)
24. Jahanian, M.H., Amin, F., Jahangir, A.H.: Analysis of TESLA protocol in vehicular ad hoc networks using timed colored Petri nets. In: 6th Int. Conf. on the Information and Communication Systems, 222–227 (2015)
25. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer, 9(3-4), 213–254 (2007)
26. Börger, E.: Modeling Distributed Algorithms by Abstract State Machines Compared to Petri Nets. In: 5th ABZ International Conference, 3{34 (2016).
27. Reisig, W.: The Expressive Power of Abstract State Machines. Computing and Informatics, 22, pp. 209-219 (2003)
28. Dershowitz, N.: The Generic Model of Computation. Electronic Proceedings in Theoretical Computer Science (2013)
29. Gurevich, Y.: Sequential Abstract State Machines Capture Sequential Algorithms. ACM Transactions on Computational Logic, 1(1), 77-111 (2000)
30. Blass, A., Gurevich, Y.: Abstract State Machines Capture Parallel Algorithms. ACM Transactions on Computational Logic, 4(4), 578- 651 (2003)
31. Glausch, A., Reisig, W.: An ASM-Characterization of a Class of Distributed Algorithms. Abrial, J.R., Glässer, U., eds., Rigorous Methods for Software Construction and Analysis, pp. 50-64 (2009)
32. Börger, E., Schewe, K.D.: Concurrent abstract state machines. Acta Informatica, 53(5), 469-492 (2016)
33. Sun, B., Guan, Y., Chen, J., Pooch, U.W.: Detecting Blackhole attack in Mobile Ad Hoc Networks. In: 5th Conf. on Personal Mobile Communications, 490–495 (2003)
34. Al-Shurman, M., Yoo, S.M., Park, S.: Black Hole Attack in Mobile Ad Hoc Network. In: 42nd Annual Southeast Regional Conference, 96–97 (2004)
35. Tamilselvan, L., Sankaranarayanan, V.: Prevention of Blackhole Attack in MANET. In: 2nd Int. Conf. on Wireless Broadband and Ultra Wideband Communications, 21 (2007)
36. Raj, P.N., Swadas, P.B.: DPRAODV: A Dynamic Learning System Against Blackhole Attack in AODV based MANET. International Journal of Computer Science Issues, 2, 54–59 (2009)
37. Ramaswamy, S., Fu, H., Sreekantaradhya, M., Dixon, J., Nygard, K.: Prevention of Cooperative Black Hole Attack in Wireless Ad Hoc Networks. In: International Conference on Wireless Networks (2003)
38. Tamilselvan, L., Sankaranarayanan, V.: Prevention of Co-operative Black Hole Attack in MANET. Journal of Networks, 3(5), 13–20 (2008)
39. Perkins, C.E., Belding-Royer, E., Das, S.R.: Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, http://tools.ietf.org/html/rfc3561 (2003)