An EFSM-based intrusion detection system for ad hoc networks

Jean-Marie Orset, Baptiste Alcalde, and Ana Cavalli

Institut National des Télécommunications GET-INT, Evry, France
{jean-marie.orset, baptiste.alcalde, ana.cavalli}@int-evry.fr

Abstract. Mobile ad hoc networks offer very interesting perspectives in wireless communications due to their easy deployment and their growing performances. However, due to their inherent characteristics of open medium, very dynamic topology, lack of infrastructure and lack of centralized management authority, MANET present serious vulnerabilities to security attacks. In this paper, we propose an intrusion detection scheme based on extended finite state machines (EFSM). We provide a formal specification of the correct behavior of the routing protocol and by the means of a backward checking algorithm, detect run-time violations of the implementation. We choose the standard proactive routing protocol OLSR as a case study and show that our approach allows to detect several kinds of attacks as well as conformance anomalies.

1 Introduction

In the recent years, one could assist to a spectacular growth in the use of wireless equipments. The number of mobile devices such as PDAs, mobile phones, laptops, is also tremendously increasing. To ensure the connectivity between all these devices, ad hoc networks appear to be a promising solution. An ad hoc network is a collection of wireless mobile nodes which communicate together without the assistance of any fixed nor central infrastructure. Thus, participants must cooperate by acting as routers and forward messages to other nodes that are not within the same radio range. MANET can be used in scenarios where no infrastructure exists, or where the existing infrastructure does not meet application requirements for different reasons such as security, cost or quality. They also open new fields of applications in the domain of networking like battlefield operations, sensor networks, emergency rescues or roaming networking.

Beside their enticing capabilities, ad hoc networks suffer from a great weakness: due to their characteristics, they are much more vulnerable than wired networks. Indeed, they rely on a open medium and have a very dynamically changing topology. They trust others participants, can not rely on a fixed infrastructure to monitor the activity, to distribute keys or to manage security policies. As a result, MANET are vulnerable to many kinds of attacks like passive eavesdropping, usurpation, routing disruption or denials of service (DoS). Recently, many schemes have been proposed to secure the routing process in MANET. Most of them are preventive: they rely on cryptographic mechanisms

for example, to authenticate participants within the network. Although they present interesting potentials, they are often inspired by the techniques used in traditional networks and are also not always well adapted to MANET (problem to exchange keys, high resource consumption, etc...). Furthermore, they only deal with *limited aspects of the protocols*. Thus, they do not allow to cope with all threats (e.g. corrupted node revealing the key, wrong messages flooding, non participation, etc...). Cryptographic mechanisms may help to identify the originators of an attack but if the attack is not detected, they remain useless. That is why one not only needs to prevent attacks but also to detect the incorrect behaviors in real time, in order to offer the network the opportunity to react efficiently. This is the role of an intrusion detection system (IDS).

Many intrusion detection schemes have been proposed in wired networks. However, the totally distributed context of MANET obliges to work out new approaches. In addition, experience shows that protocols are often subjects to design flaws which could further be exploited by an attacker to compromise the network. The contribution we bring in this paper is the definition and application of a specification-based approach (EFSM) to detect as well the anomalies in the implementation of the protocol as attacks against the routing operation. We chose the proactive ad hoc routing protocol OLSR as a case study. We thus manually abstract a correct behavior of an OLSR node according to the RFC [4]. Then, an algorithm analyzes all the messages exchanged locally by the nodes and test the conformance to the specification in real time. By the following, attacks are assimilated to violations of the specification. We illustrate that our approach allows to detect several kinds of attacks, without requiring to maintain an heavy scenario base on each nodes.

The remainder of the paper is organized as follows: section 2 describes related works on security and intrusion detection in ad hoc networks. section 3 presents the OLSR protocol as well as an overview of different attacks against ad hoc routing protocols. section 4 detailed our specification based approach and the passive testing algorithm used to perform the validation. Examples of application are described in section 3. Finally, we present the future work and conclude the paper in section 6.

2 Related work

2.1 Cryptographic schemes

In the recent years, most of the propositions to secure the routing process in MANET make use of cryptographic mechanisms. In [5] for example, the authors proposed the use of asymmetric cryptography to secure on-demand ad hoc network routing protocols. However, nodes in an ad hoc network may not have sufficient resources to verify asymmetric signatures. Thus, an attacker can trivially flood a victim with packets containing invalid signatures. Although those packets will be discarded, the verification will be prohibitively expensive for the victim.

As for the OLSR protocol, the main contribution comes from [1]. The authors proposed to rely on asymmetric cryptography to authenticate the originators of messages but also on timestamp to verify their freshness and counter replay attacks. In addition to the preceding drawbacks, there remains the problem of key certification. Since there is no fixed infrastructure in MANET, nodes can not rely on any certification authority to validate or revoke certificates.

The problem of these approaches is that even if the schemes are efficient and correctly implemented, they do not allow to cope with compromised nodes, nor authenticated nodes which deflect the normal operation of the routing process. Moreover, the secure protocol may itself contains flaws which could be further exploited by an attacker.

That is why many people propose to use intrusion detection schemes, as a complementary protection, to detect all malicious behaviors.

2.2 IDS in ad hoc networks

One of the first worth proposals to develop IDS capabilities for MANET was described in [8]. This paper presented a set of rules that identify several well known attacks on MANET. Then, they developed a cluster-based detection approach in which a node is chosen to perform detection functions for all nodes within a cluster. Although interesting, the cluster based concept induces some security problems which need to be addressed.

One of the main drawback of all statistical based IDS is the high rate of false alarms. To circumvent that problem, others people choose to rely on specification based approaches to reduce the error margin in the detection of attacks.

2.3 Specification based IDS

Some researchers [7] proposed recently a solution based on an FSM-based specification to detect vulnerabilities in the AODV protocol. They used an FSM to specify an AODV correct behavior and distributed networks monitor to detect run-time violations on the specification. In addition to the fact that FSMs constitute a too limited tool to manage the complexity of an ad hoc routing protocol, some of the assumptions are too strong so that security aspects of the architecture are not clearly addressed. Finally, in [3], the authors propose to extend the preceding solution on several aspects. Firstly, they rely on EFSA to specify the behavior of the AODV protocol. Then, they propose to use specification-based and statistical approaches complementarily, in order to detect attacks that do not directly violate the specification.

3 Attacks in ad hoc networks

3.1 The Optimized Link State Routing Protocol

The OLSR protocol is a link-state proactive protocol, based on the Open Shortest Path First (OSPF) protocol and designed specifically for mobile ad-hoc networks.

OLSR manages to diffuse routing information through an efficient flooding technique. The key innovation of this protocol is the concept of Multi Point Relays (MPRs). A node's multipoint relay is a subset of its neighbors whose combined radio range covers all nodes two hops away. In order for a node to determine its minimum multipoint relay set based on its two-hop topology, periodic broadcasts are required. Similar to conventional link-state protocols the link information updates are propagated throughout the network. However in OLSR, when a node has to forward a link update it only forwards it to its MPR set of nodes. Finally, the distribution of topological information is realized with the use of periodic topology control messages and as a result, each node knowing a partial graph of the topology of the network that is further used to calculate the optimal routes. OLSR is mostly preferred when the ad hoc network consists of a large number of nodes and has a high density. One of the main advantages of the OLSR protocol is that it does not make any assumption concerning the underlying link layer, allowing it to be used in a variety of configurations.

3.2 Vulnerabilities

Currently, OLSR does not specify any special security measures. As a proactive routing protocol, OLSR makes a target for various attacks. In OLSR, each node is injecting topological information into the network through the transmission of *HELLO* messages and, for some nodes, *TC* messages. If some nodes (malicious or malfunctioning) inject invalid control traffic, network integrity may be compromised. Here are examples of situations that may occur due to lack of data integrity functionality.

Identity usurpation By sending false *HELLO* messages, a node may pretend to be another node. Then, the target node identifies the originator address as from one of its neighbor. This may result in creating conflicting routes to the node with the corresponding address, errors in the routing tables, link loss, etc...

Insertion of false control messages A node broadcasts a wrong *HELLO* message claiming symmetrical links to non-neighbor nodes or to non-present nodes. As a consequence, the node may be selected as an MPR and the traffic between other nodes will be routed by itself. It can then discards all the traffic or just the control messages (*Hello*, *TC*) to disturb the routing operation.

Alteration of control messages A node may listen to the *TC* messages from neighbors, add non existing nodes with symmetrical connectivity and replay the packet by spoofing the originator. Then, the target node is designed as an MPR whereas it is unable to reach the nodes. In the same manner, a node may also include non-existing links (i.e. links to non-neighbor nodes) in a *TC* message. That may yield routing loops and conflicting routes in the network.

Another alteration attack which has a great impact consists for an attacker, to forward a TC message with a sequence number increased from x to $x + i$. By

the following, the receiver will stop to analyze and forwards packets from the originator with sequence number lower than $x + i$.

As we can see, OLSR present several serious vulnerabilities which could lead to paralyze the communications within the network. All the cited attacks hijack the normal operation of the protocol and can not always be prevented by classic authentication schemes. Indeed, such schemes only allow to verify the origin of messages but not their relevance. By formally specifying the normal operation of OLSR, we are able to detect many anomalies and intrusions on the protocol since each one will be detected in real-time, as a violation of the specification. Moreover, it is very difficult in MANET to establish a clear difference between a malicious behavior and normal operations in some circumstances (battery sparing, presence of obstacles, very dynamic topology, etc...). By using a formal specification, we allow to decrease consequently the margin of error in the detection of attackers.

4 Description of the IDS

4.1 Motivation and assumptions

Intrusion detection techniques used in traditional (wired) networks can not be applied directly to ad hoc networks. In wired architectures, the monitoring is generally realized on routers or hubs. In ad hoc networks, there is no infrastructure nor single management entity, making it difficult to perform any kind of centralized management or control. Hence, the intrusion detection has to be distributed to all nodes within the network. We also achieve the distribution of the intrusion detection mechanism by implementing a Local Intrusion Detection System (LIDS) on each node. A potential drawback of this approach could reside in the difficulty to correlate the traces of the different nodes. Indeed, if one states that every node is an omnipotent observer that can see and monitor all the traffic inside the network, one has to rely on strong assumptions such as a global synchronization of nodes (to order the different traces) and a strong authentication mechanism to ensure the integrity of the traces exchanged by the different nodes. However in our approach, such complex mechanisms are not required since one only needs to analyse the informations exchanged between nodes of the same neighborhood. Indeed, the use of MPR relays in OLSR allows to divide the network in different subsets. Thus, each node only needs to know the messages locally, inside the MPR sets.

Considering the specification, our algorithm allows to detect errors in an exhaustive manner. Thus, we are able to detect violations which correspond to conformance errors as well as violations which directly result from a security attack. To characterize more precisely the real nature of the detected violation (attack or error in the implementation itself) one may use a complementary intrusion detection scheme based on attack signatures. Accordingly, each time an anomaly is raised, an algorithm is applied to check if this corresponds to the

pattern of any well-known attack. In that case, our approach is similar to misuse detection schemes since it allows to efficiently detect instances of known attacks. On the other hand, if the anomaly does not corresponds to any attack pattern, it may either imply that one identified a new attack, either that one raised an error in the implementation itself, what anyhow characterizes a potential flaw in the routing protocol. Afterward, this analysis has to be performed by an expert of this domain or by a dedicated algorithm. This is out of the scope of this paper.

What is important to keep in mind is that our approach allows to detect truly innovative errors, compared to signature based schemes which are only able to detect well known attacks. Another significant advantage is that there is no false positive since one only considers the normal behavior described in the specification. Thus, whatever the anomaly the algorithm raised, one can always be sure it corresponds to a potential flaw.

4.2 Extended Finite State Machine

We chose to rely on the EFSM formalism because it suits very well to the analysis of flows and allows to put constraints on the variables of the transitions.

Definition 1. *An Extended Finite State Machine M is a 6-tuple $M = < S, s_0, I, O, \vec{x}, T >$ where S is a finite set of states, s_0 is the initial state, I is a finite set of input symbols (eventually with parameters), O is a finite set of output symbols (eventually with parameters), $\vec{x}$ is a vector denoting a finite set of variables, and T is a finite set of transitions. A transition t is a 6-tuple $t =< s_i, s_f, i, o, P, A >$ where s_i and s_f are the initial and final state of the transition, i and o are the input and the output, P is the predicate (a boolean expression), and A is the ordered set (sequence) of actions.*

We manually derived the EFSM directly from the IETF specification [4]. The verification process consists to map the traces of I/O events (messages received and sent) recorded on each node, with the specification. To compare the traces with the EFSM, we chose an approach based on backward tracking [2].

Another consequent advantage brought by the backward testing approach is that it allows to consider that the traces can start at any moment of the implementation execution, not only the initial state.

Given a trace from the implementation, our algorithm will detect three types of errors:

- **the output errors** : when the output of a transition in the implementation differs from the output of the corresponding transition in the specification.
- **the transfer errors** : when the ending state of a transition in the implementation differs from the ending state of the corresponding transition in the specification.
- **the mixed errors** : a mix between the two errors defined above.

4.3 Overview of the Backward Checking Approach

The Backward Checking algorithm is an approach of passive testing on EFSMs derived from the testing by determination of variables intervals. We consider that we have a system under test on which we place an observation point. We suppose that this observation point records the event traces respecting their causal order. We assume that finding the order of these events is a well studied and resolved problem.

The Backward Checking algorithm (cf. [2]) is composed of two main phases. First, it follows a given event trace backward to find the possible initial configurations at the beginning of the trace.
Secondly, it starts from these configurations and explore backward, every possible path of the specification with help of pruning operation and a transition choice strategy, to reduce as much as possible the search.

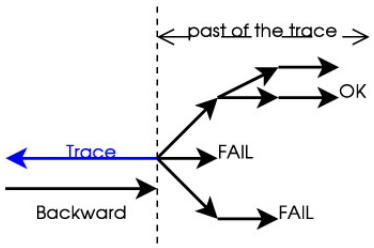

Fig. 1. Overview of Backward Checking

In the trace During the first phase, if an inconsistency is detected it means that the event trace is not correct and that an error has been detected. If at the end of the event trace analysis no inconsistency has been found, it means the studied trace is possible with the obtained initial configuration. Then, the second phase is launched, i.e. the exploration of the past of this trace.

In the past of the trace In the second phase, we try to confirm the intervals in which the variables are defined according to the initial configurations. The algorithm finishes with a positive answer (trace is valid) at the first confirmed configuration, i.e. the first configuration in which every determinant variables has been confirmed, or with a negative answer (invalid trace) if every branch of the exploration tree leads to an inconsistency. One can say that this algorithm is optimal in the way that it will be fast to say that there is no error, but slower

to say that there is one. This fact is coherent with reality because we suppose that an error in an event trace is an exceptional behavior.
The different branches of the exploration tree are the possible successions of transitions, taken in a backward manner from the initial configurations resulting from the first phase.

4.4 OLSR Extended Finite State Machine

For the sake of simplicity, we decided not to specify all the functionalities of OLSR. For example, we consider that each node has only one interface and also that it can only claims one link to a same node. Since OLSR is a link state routing protocol, we decided to model the behavior of every node according to its state and its connectivity with its neighbors. Indeed, OLSR makes difference between links depending if they are asymmetric or bidirectional and also between nodes, depending if they are normal nodes or multipoint relays - what implies they have a priori, a better connectivity. Our EFSM only represents the interactions between two nodes. This implies there is a unique EFSM for each link between two nodes. We reuse the notation of [3], i.e. we use the abbreviation *obs* and *cur* to specify respectively the destination and the local node (*obs* stands for **obs**erved node and *cur* for **cur**rent node). The events which correspond to a received packet are noted with a '?' while those with a '!' relate sent packets. The typical notation is also "*address - type of event - type of message - fields*".

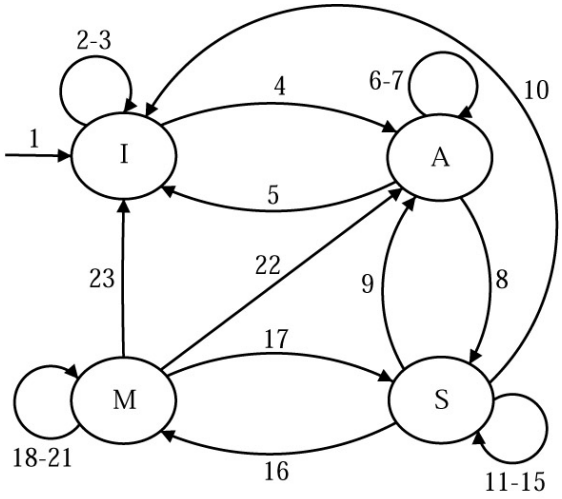

Fig. 2. OLSR Extended Finite State Machine (see Appendix for details)

The goal of the first transition is to initialize all the timers and variables. OLSR is a proactive protocol, so it needs to periodically send control messages

though the network to keep the link states updated. This is represented on the figure by the timers *HelloTimer* and *UpdateTimer* which respectively time the *Hello* message sending and the the link state updating. Normally, a node should update its link state as soon as it receives a *Hello* message. However, we needed two timers here, to specify the behavior of a node when he receives no message (its connectivity may decrease). Thus, in the ***I****nitial* state, the node uses the first timer to perform its neighbor discovery. Since it has none actually, it sends an empty *Hello*. It remains in the same state if it has received no message after the specified second timeout.

By receiving a message from another participant, the node immediately goes on the second state ***A****symmetrical*, which means it has heard another node but can not yet claim a symmetrical link with it. We also record the address of the heard node in a list of asymmetrical neighbors to further verify if one can rightfully pass in the next state. From that state, a node may:
- receive no message and return in the preceding state after the timeout
- receive another *Hello* message from the same node and remain in that state
- receive an *Hello* message from a node

If the sending node has recorded it as an asymmetrical one (use of the constant $ASYM$), one can pass in the ***S****ymmetrical* state. But this is only plausible if the current node previously sent an empty *Hello* message, what can be verified by the mean of the variable *SentHello*, which must be set to *true*. This precaution allows to avoid that malicious nodes claim non-existent links. The sending node could also have directly recorded it as a symmetrical neighbor (use of constant SYM). In that case, that node must have been recorded previously in the asymmetrical list($AsymList$).

Once it reaches the symmetrical state a node has to advertise its neighbors. Hence, it may:

- send a Hello message to claim a symmetrical link with the observed node
- advertise that it chose the observed node as one of its MPR nodes (by using the constant *MPR*)
- send data to one of its MPR, to be forwarded
- receive ***T****opology* ***C****ontrol* messages (only if the sender has been previously chosen as a MPR)

As for the receiving, if the node only receives empty *Hello* after a certain time (it is not seen anymore), it goes back in the asymmetrical state, since it can still listen nodes. If it is claimed by the observed node as an asymmetrical or a symmetrical node, it stays in its state. Finally, the node can be chosen as a MPR (the constant *MPR* is used). If so, it records the address of the sender in its MPR selectors list ($MprSelList$).

When a node finally reaches the MPR state, he has to keep on advertising periodically its MPR selectors. From now, it has also the responsibility to forward data from selectors. To do that, it just has to ensure that the message really comes from one of its MPR selectors, what can be easily verified by examining the list *MprSelList*. The second function of MPR nodes is to generate the link state messages. The representation we choose allows easily to verify that this kind of message can only be sent by an MPR node, as specified in [4]. Finally, depending on the received messages, the node will eventually reach another state. If it keeps on being claimed as a MPR node by the same neighbor, it stays in that state. If it is claimed only as a symmetrical node, it will become again a simple symmetrical neighbor, regarding the sender. As for the preceding state, if it receives message in which it can find its own address, it goes back to the asymmetrical state. Finally, if after the timer expires it has not received any message, it goes back to the initial state. In both cases, the lists are flushed and the timer are reset.

One has to note that there is an EFSM for each connection between two participants. Thus, a node could be in the *MPR* state regarding one node and be in a *SYM* or *ASYM* state with relation to another one.

5 Detecting Attacks

To test the effectiveness of our approach, we applied our violation detection algorithm to the traces corresponding to the two attacks described in section 3.

5.1 Hello message insertion

Let us recall that in this scheme, an attacker advertised a non-existent symmetrical link to its neighbors, thus perturbing their routing table calculation process. According to our notation (see Appendix), the corresponding trace is also:

```
HelloTimerOut / cur!Hello()
UpdateTimerOut
obs?Hello()
UpdateTimer
obs?Hello(cur) with cur=SYM
```

Here is how our algorithm proceeds to verify the trace: starting from the last event, it determines which could be the corresponding transitions. By examining the different possible transitions, one observes that this event can match the transitions 8, 15 and 17, which respectively start from the states A, S and M. These are the only transitions which correspond to the reception of a *Hello* message claiming a symmetrical link. Here, we have also three state possibilities with the following parameters:

State: A; Parameters: cur=SYM, obs∈AsymList
State: S; Parameters: cur=SYM
State: M; Parameters: cur=SYM, obs∈MprList

At that time, the algorithm shifts on the preceding event (*UpdateTimer*) and searches within the EFSM, all the transitions containing that event, which can lead to the states *A*, *S* or *M*. By looking into the list of all transitions, one can immediately verify that there is none which satisfy the requirements. The algorithm also raises a transfer error violation, that reveals the trace does not corresponds to the specification. From this anomaly, one may bring out two conclusions. Either we consider the implementation is correct and thus, there is a security attack in the network, either one considers that the implementation needs to be validated and we also exhibit a conformance error.

5.2 MPR usurpation

Let us now consider the second attack in which an attacker sends Topology Control messages without having been elected as a MPR. A corresponding trace would have the pattern:

```
cur!Hello(obs) with obs=MPR
cur!Hello(obs) with obs=SYM
obs!TC(cur) with cur=MPRSEL
```

This is an example of trace where a node keeps on claiming to be a MPR node after having been demoted as a simple symmetrical one. The current node elects the observed one as a MPR (and may perform some transitions). Then, it demotes the observed node (after having recalculated its routing table for example) but keeps on receiving *TC* messages from it.

The process is then the same than for the previous attack. The algorithm searched for a transition which corresponds to the emission of a *TC* message with the variable '*cur*' set to '*MPRSEL*'. It appears that this event match the transition 14. This transition leads to state with the predicate *obs∈MprList*. So now, the algorithm shifts to the previous event (*cur!Hello(obs) with obs=SYM*) and searches backward all the transitions that contained it, which also reach the state *A*. By checking all the possible transitions in the EFSM, it appears that only the transition 11 matches. Nevertheless, the transition 11 has, as an effect, to remove the address of the observed node from the *MprList* of the current node. That is in total opposition with the predicate of the transition 14. The algorithm also raises this time, an output error and discloses a new anomaly. Again, it can result from a dysfunction in the observed node or a willful security attack.

Note that this scheme is quite similar to the case where a node falsely advertises symmetrical links through the *TC* messages. Both anomalies will be identically detected.

Thus, it appears that our approach allows to detect several kinds of typical attacks on the OLSR protocol. These attacks are not taken into account in the original specification of the protocol and could constitute a serious risk if one is not able to detect them.

6 Conclusions and further work

We proposed a specification based approach that rely on the use of extended finite state machines to detect attacks on the OLSR protocol. The use of EFSM makes possible to analyze in depth, the messages exchanged between nodes. We applied a backward checking algorithm to detect violations on the specification. This approach brings a significant benefit on the quickness of the verification process, what is crucial in the context of run-time verification. We then, applied our algorithm to detect flaws on the OLSR protocol and showed that it makes it possible do detect several kinds of anomalies.

We plan to integrate this approach and this algorithm in a complete IDS infrastructure. We also envisage to use in a complementary way, a signature analysis tool to detect attacks that can not yet be easily detected by a specification-based approach(e.g. DoS. attacks).

References

1. Cédric Adjih, Thomas Clausen, Philippe Jacquet, Anis Laouiti, Paul Mühlethaler, and Daniele Raffo. Securing the olsr protocol. In *Proceedings of IFIP Med-Hoc-Ned*, pages 125–134, 2003.
2. B. Alcalde, A. Cavalli, D. Chen, D. Khuu, and D. Lee. Network protocol system passive testing for fault management - a backward checking approach. In *Formal Techniques for Networks and Distributed Systems - FORTE 2004*, pages 150–166, Madrid, Spain, september 27-30 2004. Springer.
3. Yi an Huang and Wenke Lee. Attack analysis and detection for ad hoc routing protocols. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID'04)*, 2004.
4. Thomas Clausen and Phillipe Jacquet. *IETF RFC 3626: Optimized Link State Routing Protocol (OLSR)*. The Internet Society `http://www.ietf.org/rfc/rfc3626.txt`, 2003.
5. B. Dahill, B. Levine, E. Royer, and C. Shields. A secure routing protocol for ad hoc networks, 2001.
6. Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking MobiCom 2002*, pages 12–23, 2002.
7. Chin-Yang Tseng, Poornima Balasubramanyam, Calvin Ko, Rattapon Limprasittiporn, Jeff Rowe, and Karl Levitt. A specification-based intrusion detection system for aodv. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 125–134, New York, NY, USA, 2003. ACM Press.
8. Yongguang Zhang, Wenke Lee, and Yi-An Huang. Intrusion detection techniques for mobile wireless networks. *Wirel. Netw.*, 9(5):545–556, 2003.

Appendix

Here are the different transitions corresponding to the EFSM of the figure 4.4. P are used to denote predicates while A stand for actions. Uppercase variables denote constants and are used to specify the nature of a node depending if it is asymmetrical, symmetrical, a MPR node or a MPR selector.

1. A: reset SentHello, HelloTimer, UpdateTimer, TcTimer; clear AsymList, MprSelList, MprList

2. HelloTimerOut / cur!Hello()
   A: set SentHello=true; set HelloTimer

3. UpdateTimerOut
   A: reset UpdateTimer ; set SentHello=false

4. obs?Hello()
   A: reset UpdateTimer ; Add(obs,AsymList)

5. UpdateTimerOut
   A: reset UpdateTimer; Remove(obs,AsymList)

6. HelloTimerOut / cur!Hello(obs)
   A: set obs=ASYM; reset HelloTimer

7. obs?Hello()
   A: reset UpdateTimer

8. obs?Hello(cur)
   P: (cur=ASYM AND SentHello=true) OR (cur=SYM AND obs∈AsymList)
   A: reset UpdateTimer

9. obs?Hello()
   A: reset UpdateTimer; reset SentHello; remove (obs,MprList)

10. UpdateTimerOut
    A: reset UpdateTimer; Remove(obs,AsymList); reset SentHello; remove(obs,MprList)

11. HelloTimerOut / cur!Hello(obs)
    A: set obs=SYM; reset HelloTimer; remove(obs,MprList)

12. HelloTimerOut / cur!Hello(obs)
    A: set obs=MPR; reset HelloTimer; add(obs,MprList)

13. cur!Data()
    P: obs∈MprList

14. obs?TC(cur)
    P: cur=MPRSEL AND obs∈MprList

15. obs?Hello(cur)
    P: cur=SYM OR cur=ASYM
    A: reset UpdateTimer

16. obs?Hello(cur)
    P: cur=MPR
    A: Add(obs,MprSelList); reset UpdateTimer; reset TcTimer

17. obs?Hello(cur)
    P: cur=SYM OR cur=ASYM
    A: reset UpdateTimer; Remove(obs,MprSelList)

18. HelloTimerOut / cur!Hello(obs)
    A: set obs=MPRSEL; reset HelloTimer

19. TcTimerOut / cur!TC(obs=MPRSEL)
    A: reset TcTimer

20. obs?Data() / cur!Data()
    P: obs∈MprSelList

21. obs?Hello(cur)
    P: obs∈MprSelList AND cur=MPR
    A: reset UpdateTimer

22. obs?Hello()
    A: reset UpdateTimer; clear MprSelList

23. UpdateTimerOut
    A: reset updateTimer; clear AsymList; clear MprSelList; reset SentHello