Using Symbolic Techniques and Algebraic Petri Nets to Model Check Security Protocols for Ad Hoc Networks

Mihai Lica Pura and Didier Buchs

Centre Universitaire d'Informatique
University of Geneva
Carouge, Switzerland

Abstract. Petri nets have proved their effectiveness in modeling and formal verification of a large number of applications: control systems, communication protocols, application workflows, hardware design, etc. In the present days, one important focus of computer science is on security and secure communications. The use of Petri nets for verifying security properties is not a mature field due to a lack of convenient modeling and verification capabilities. So far, in the Petri Net field there is only the CPN tool that is mature enough for modeling using the colored Petri nets formalism. Nevertheless verification cannot be performed on large systems as CPN tool verification is based on an exhaustive way of computing the semantics of a model. In this paper we present the use of AlPiNA, another candidate for this task. AlPiNA is a symbolic model checker that uses the formalism of algebraic Petri nets. We have used it successfully for modeling ad hoc networks and for verifying security protocols designed for this type of networks. As a case study and benchmark we have chosen the ARAN secure routing protocol. We managed to find all the attacks that were already reported for this protocol. To our knowledge this work is also the first successful attempt to use Petri nets for model checking the security properties of ad hoc networks protocols.

Keywords: model checking, ad hoc networks, algebraic Petri Nets.

1 Introduction

Place/Transition nets are a modeling language that proved its effectiveness in modeling a large variety of systems based on concurrent processes. Over the years, the initial Petri net formalism was enriched in order to simplify the specification of more and more complex systems. Two of the applications targeted were the model checking of security protocols and of the ad hoc network protocols (but not ad hoc network security protocols). To the best of our knowledge, model checking the security protocols specially designed for ad hoc networks has not been reported yet.

There is no need to argue for the importance of security in computer science, or for the need to prove the security properties of the protocols used in the information systems. Ad hoc networks are a novel approach to assuring communications. The communications networks that are now in use are based on an infrastructure composed of devices like switches, hubs, gateways, routers, and so on. Ad hoc networks aim to assure communications without the use of any infrastructure. In such networks there are no other devices, except the ones that actually form it, and want to communicate. And they will also act as the infrastructure devices from a classical network, by routing the messages of all the other nodes. Such a behavior is assured by specially designed ad hoc routing protocols. These routing protocols and their possible attack schemes are more complex than the ones of the other kinds of networks, so for their specification a more powerful language is needed.

One of the enrichments of P/T nets dedicated specifically to data based functionality is High Level Petri Nets (HLPN). In HLPN the tokens have different types and these types are part of a many-sorted algebra ([1]). The possibility to use other types than the usual black tokens made it possible to use HLPN in modeling and verification of security protocols.

Colored Petri Nets (CPN) were the first concrete realization of HLPN that were used for model checking security properties, because they were the first one who was expressive enough for this ([2]). But besides CPN, there are other implementations of HLPN. The difference between the different implementations of HLPN stands in the way the many-sorted algebra is defined. In CPN the many-sorted algebra is defined using the CPN ML language, which was built upon the standard ML.

For modeling ad hoc networks we focus on the model checker AlPiNA ([3, 4]). AlPiNA implements HLPN by algebraic Petri nets (APN), in which the colored tokens are defined using algebraic abstract data types (AADT) ([1]). Like all the other model checkers, the focus of AlPiNA is to handle the state explosion problem in order to perform verification on real size system models. When using HLPN, the state space explosion has one more dimension (the data) than in the case of P/T nets. HLPN are more expressive and as a consequence, the state space of a HLPN model is in general much bigger. AlPiNA addresses this problem by using symbolic techniques based on several layers of Data Decision Diagrams, Set Decision Diagrams and Sigma Decision Diagrams [1]. In addition, some optimizations specific to the APN formalism (algebraic clustering, partial algebraic unfolding) [5] are supported. The tool can be downloaded from [5].

We have successfully used AlPiNA for modeling ad hoc networks and for model checking security protocols of ad hoc networks. From our studies, we have seen some advantages that this tool has over the other tools used for these purposes; in terms of modeling the protocol itself, as well as the possible attackers. In this paper we will present the modeling of ad hoc networks and the verification of ARAN (Authenticated Routing for Ad Hoc Networks [6]) security protocol with APNs, and the advantages of AlPiNA for performing these tasks.

The rest of the paper is organized as follows. The second section presents the use of Petri nets in literature for modeling ad hoc networks and verifying properties related to them. In the third section we describe the use of algebraic Petri nets and AlPiNA for modeling ad hoc networks and the ARAN protocol. The fourth section contains the presentation of our results regarding verification of routing information correctness for ARAN. The last section contains conclusions and our future work directions.

2 The Use of Petri nets in modeling ad hoc networks

Petri nets already proved their effectiveness in modeling ad hoc networks. So far, researchers have used Fuzzy Petri nets, Stochastic Petri nets and Colored Petri nets to model ad hoc networks. The purpose of these models was to obtain qualitative or quantitative information about the behavior of applications and protocols in the context of ad hoc networks. As far as we know, algebraic Petri nets were never used so far to model ad hoc networks.

We will continue by presenting some of the latest published results concerning the use of Petri nets in ad hoc networks research.

2.1 Modeling for Quantitative evaluation

The research presented in [7] uses Fuzzy Petri Nets for modeling and analyzing the QoS dimension in order to evaluate how to manage congestion in wireless ad hoc networks. The networks itself, the nodes, the communication protocol are not actually modeled. In [8] Fuzzy Petri Nets are used to represent the multicast routing in an ad hoc network and to calculate multicast trees. The authors only model the topology of the network but not the actual routing protocol.

In [9] the authors present how to use Stochastic Petri Nets to model ad hoc networks. An ad hoc network is modeled by a single node, for which a proper amount of traffic is generated. By measuring how the node behaves under the given traffic, using suitable metrics, some conclusions can be obtained regarding a whole network with a given number of nodes like the modeled one. In [10] Stochastic Petri Nets are used to model mobility of ad hoc networks, but the actual ad hoc network is not modeled, neither the ad hoc routing, only an application level protocol that takes into account the fact that the nodes are moving between different geographic regions, and also the required performance indices. Thus the authors are able to obtain quantitative data about the specified performance indices.

The authors of [11] and [12] use Colored Petri Nets. They propose models for the nodes of the network, for the routing protocol AODV (Ad Hoc On-Demand Distance Vector Routing) [12] and DSR (Dynamic Source Routing) [11] and for the behavior of the ad hoc network. The purpose of the modeling was to conduct a comparison between the two ad hoc routing protocols mentioned above, from the point of view of their efficiency (number of generated overhead packets, data packet delivery delay). In [13] Colored Petri Nets are used to model and to compare another pair of routing protocols, AOMDV (Ad Hoc On-Demand Multipath Distance Vector Routing) and DSR. In [14], Colored Petri Nets are used to model and validate the specification of a multicast routing protocol for ad hoc networks called DYMO (Dynamic MANET On-Demand). The properties that the authors specify and verify are all related to the correctness of the protocol: establishments of routes, and correct processing of the routing messages. By this work, the authors also found several ambiguities in the definition of the protocol, which were taken into consideration in two revisions.

2.2 Modeling for Qualitative evaluation

From the point of view of model checking security protocols, Colored Petri nets are the only type of Petri nets used for this purpose up to now. But as far as we know, no Petri nets were used to model check the security protocols of ad hoc networks. So our paper is the first presentation using algebraic Petri nets to model ad hoc networks and to do model checking of security properties for specific ad hoc network protocols.

For example, [2] and [15] present the work of using CPN to model check confidentiality and authentication for TMN authenticated key exchange protocol. In [16] CPN are used to verify the same security properties for Andrew secure RPC protocol. In all these papers, the use of CPN helps to find attacks over the considered protocols, and even some attacks that were previously unknown. So this indicates the high potential of using these techniques for model checking ad hoc network specific security protocols.

In the next sections, we will present the state of the art of modeling ad hoc networks with the help of Petri nets. Modeling an ad hoc network implies modeling the following elements: the nodes and the topology of the network.

2.3 Modeling the nodes

For modeling the nodes of an ad hoc network, a single approach was used by all the researchers. The nodes were modeled by their behavior in the considered protocol or application. The Petri net contains a single instance of a node's behavior. But this behavior is parameterized with the identity of a node. The identities of the nodes, which are part of the considered network, are placed inside a special place. When the state space is calculated, all these identities are considered as executing the modeled behavior ([11]).

2.4 Modeling the topology

When modeling the topology of the ad hoc networks, two aspects should be taken into consideration. The first one is how to model the actual topology of the network at a given time. The second aspect is how to model the mobility of the nodes which implies the modeling of the dynamicity of the topology. Both of these aspects influence the modeling of the way messages travel through the network. Based on the current topology, a message transmitted by a node should only be received by the other nodes which are in the coverage area of the transmitting node.

So far, researches have proposed three ways for modeling topology. We will briefly present them in the following paragraphs.

In [11], [12] and [13] the network topology was modeled by an approximation mechanism. Let us presume that the network has n nodes. When a node A sends a broadcast message, it actually sends n-1 copies of the message to a place that stores them in order to distribute them to the corresponding nodes. Based on a probability that represents how many nodes are in the coverage area of A, a certain number of these messages will be forwarded to other nodes, and the remaining messages will be dropped. In the case of unicast messages, they are sent only to the corresponding nodes. The authors of [12] call this model a topology approximation mechanism and prove through simulation that it can indeed mimic the mobility of a mobile ad hoc network (MANET).

In [14] the wireless mobile ad hoc network is modeled by two parts: a part that handles the transmission of the packets, and another part that handles the mobility of the nodes. The transmission of the packets is done based on the current topology of the network, which is explicitly represented in the following way: each node A has an adjacency list of nodes. Each node from this list is a node that is in the coverage area of A, and thus can receive packets from it. Based on the information from these lists, the transmission part of the model of the ad hoc network sends the packets to the appropriate nodes. The mobility part of the model is responsible with making modification to the topology. At the beginning of the validation, there is an initial topology and also the possible topology changes. Based on these changes, the mobility part modifies the topology as the validation continues.

The authors of [17] and [18] use reconfigurable algebraic higher-order net systems in order to model mobility for the ad hoc networks. The idea is to apply graph transformation (rewriting of the model) to algebraic nets. That is, the net gets reconfigured at run time in order to simulate the mobility of the nodes in an ad hoc network. The modeling is abstracted from the network layer, and the considered application is modeled in terms of work-flows.

3 Using Algebraic Petri Nets in Modeling Ad Hoc Networks

3.1 Algebraic Petri nets definition

An APN is a HLPN where algebraic abstract data types are used. The structure of the net is the structure of a Place/Transition net, but algebraic values are used as tokens. Also, the transitions can have guards that are pairs of algebraic terms that allow the firing of the respective transitions. In the following a sketch of the model components are given, more details can be found in [1].

An algebraic Petri net specification is a 5-tuple
$N - SPEC =< Spec, T, P, X, AX >$, where:

- $Spec =< \Sigma, X', E >$ is an algebraic specification extended in $< [\Sigma], X', E >$, where $[\Sigma]$ is a multiset over the signature $\Sigma =< S, F >$ ([19]) such that:

  - S is a finite set of sorts;
  - $F = (F_{w,s})_{w \in S^*, s \in S}$ is a $(S^* \times S)$ sorted set of function names;
- T is the set of transition names;
- P is the set of place names and there is a function $\tau : P \to S$ which associates a sort to each place;
- X is a S-sorted set of variables;
- AX is a set of axioms and it will be defined below.

Given an algebraic Petri net specification $N{-}SPEC = < Spec, T, P, X, AX >$, an axiom in AX is a 4-tuple $< t, Cond, In, Out >$ such that:

- $t \in T$ is the transition name for which the axiom is defined;
- $Cond \subseteq T_{\Sigma,X} \times T_{\Sigma,X}$ is a set of equalities attached to the transition name t for this axiom; $Cond$ is satisfied if and only if all the equalities from the set are satisfied;
- $In = (In_p)_{p \in P}$ is a P-sorted set of terms such that $\forall p \in P, In_p \in (T_{[\Sigma],X})_{[\tau(p)]}$ is the label of the arc from place p to transition t;
- $Out = (Out_p)_{p \in P}$ is a P-sorted set of terms such that $\forall p \in P, Out_p \in (T_{[\Sigma],X})_{[\tau(p)]}$ is the label of the arc from transition t to place p.

In AlPiNA, the input of a transition is a set that can only contain variables and closed terms [4]. However, this limitation has no effect over the complexity of the systems that can be modeled and verified. It is just simplifying the complexity of the computations.

In order to provide a semantics to a specification $N - SPEC$, we can define the set of reachable states $StN - SPEC(M)$ from a given marking M. In this paper we do not need the precise definition; please consult [1] for more details.

3.2 Case study: ARAN secure routing protocol

In order to present our methodology for modeling ad hoc networks, we have taken as case study the ARAN secure routing protocol. We have chosen it because it is simple, well known and it is the state of the art regarding secure routing in ad hoc networks. The purpose of ARAN is to provide a route path for any node in the network. It is an implicit routing protocol, which means that it will not respond with the whole path, but only with the identity of the next node in the path. ARAN uses digital signatures to assure authentication and integrity for the exchanged routing information.

ARAN uses two message types: route discovery and route response. Each message is signed by its source node. As it travels to its destination, the signed message is also cosigned by each intermediate node, after eliminating the signature of the previous intermediary, if it exists. Each node validates the received message by validating the signature(s) from the message. If the signature(s) are not valid, the message is discarded. Otherwise, the intermediary node broadcasts the message, if it is a route discovery message, or unicast it, if it is a route response message. When a route discovery message reaches destination, the node will respond with a route response message. When a route response reaches destination, the node will modify its routing table accordingly. Also, each intermediary node that receives a routes response for a route discovery that he processed, will also update its routing table. Each route from the routing table has a given lifetime. When no traffic has occurred on an existing route for that route's lifetime, the route is deactivated. When data is received for an inactive route, the corresponding note will demand the source node of the data to make a new route request for the targeted destination node. So topology changes will determine route inactivation in some nodes' routing tables, which will further determine new route requests for the destination. For more information regarding the protocol, please consult [6].

The modeling of ARAN for the purpose of its verification implies the modeling of the following elements: the nodes, the ad hoc network, the adversary and the protocol operation. The general model for ARAN is given in Fig. 1. We will now continue with the presentation of all the parts of the model.

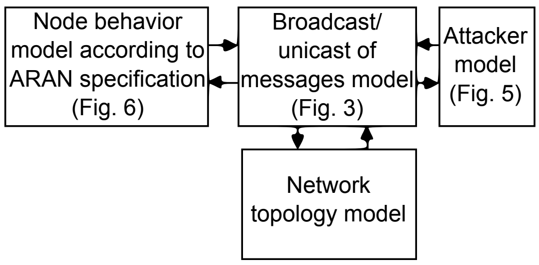
Node behavior model according to ARAN specification (Fig. 6)
Broadcast/ unicast of messages model (Fig. 3)
Attacker model (Fig. 5)
Network topology model

Fig. 1. ARAN general model

3.3 Modeling the nodes

A node of the ad hoc network is modeled as a AADT *Node*. Each node has an identity which is unique in the ad hoc network. Each node has also a routing table and some other structures needed for the operation of the considered ad hoc routing protocol. Because ARAN uses digital signatures, each node also has a pair of public/private keys and a digital certificate. In addition, each node knows the public key of the certification authority that issued his certificate.

Since all the nodes are identical, they all behave the same way. So in the actual Petri net, all the nodes are placed inside the same place called *Nodes* collecting identifiers of type *Node*. Here is the AADT *Node* in the case of ARAN:

```
Adt node
Sorts node;
Generators
node: Identity, RouteDiscoveryRequests, RouteDiscoveryRequests,
          RoutingTable, Nonce, Certificate, PrivateKey,
          PublicKey -> Node;
Operations
get_identity: Node -> Identity;
...
Axioms
get_identity(node($i, $rdr, $rp, $rt, $n, $c, $priv, $pub))=$i;
Variables
i : Identity;
...
```

All the elements used by the generator for the AADT *Node*, are other AADTs that define (in this order): the identity of the node, a list with the route discovery requests that were already broadcasted, a list with the route discovery responses that were already forwarded, a lists with the routes, the current value for the nonce used in the messages, the certificate of the node, the private key of the node, and the public key of the certification authority that issues certificates for the nodes.

3.4 Modeling the topology

An ad hoc network can be defined as a graph. We have assumed the connections are bidirectional, so the graph is an undirected one. The nodes of the graph are the nodes of the ad hoc network, and the arcs represent the fact that two nodes can communicate directly through their wireless devices. So the topology of an ad hoc network can be represented as a graph. We modeled it as the AADT *Topology*, which is in fact a list of pairs of node identities, and represents the arc list that defines the graph.

The actual topology is a variable of the type *Topology*. Its value can be given in two different ways. Depending on the type of properties that will be verified, the first or the second approach will be preferred. The first way is to give the value explicitly. In this case, the model will represent the exact ad hoc network that has that topology. For example, the topology of the ad hoc network given in Fig. 2, will be defined by the next term:

```
cons(pairIdentityIdentity(i(i0), i^2(i0)),
cons(pairIdentityIdentity(i^2(i0), i(i0)),
cons(pairIdentityIdentity(i^2(i0), i^3(i0)),
cons(pairIdentityIdentity(i^3(i0), i^2(i0)),
cons(pairIdentityIdentity(i^3(i0), i^4(i0)),
cons(pairIdentityIdentity(i^4(i0), i^3(i0)),
cons(pairIdentityIdentity(i^3(i0), i^5(i0)),
cons(pairIdentityIdentity(i^5(i0), i^3(i0)), empty))))))))
```

The second way is to not assign any value to the variable. This way it will be a free variable. Then, with the use of domain unfolding, AlPiNA will generate for that variable all the possible values within a given range. We will next explain how this works and the impact of such a choice.

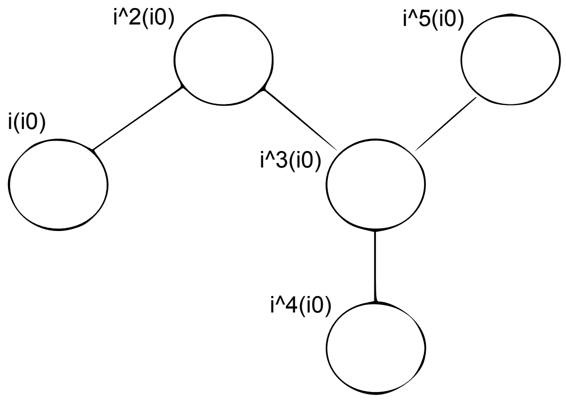

Fig. 2. An example of an ad hoc network topology

3.5 Using unfolding to model topology

Unfolding is used for the verification process in order to let the user define the part of the domain of a data type that will be taken into consideration when the state space is computed. For example, in our model, the *Identity* AADT is used for the identification of nodes. So when a certain operation must be done for all the nodes in the network, that operation is parameterized with a variable of type *Identity* for which no value is specified. Then the type *Identity* is unfolded to the number of nodes in the network. As a result, prior to building the state space, AlPiNA will unfold the Petri net by considering for that *Identity* variable all the possible values, up to the number of nodes in the network. Let us show how we can use this technique to model the topology of the ad hoc networks.

Topology AADT is actually a list of pairs of identities. Each pair of identities represents a direct connection in the ad hoc network and it is defined by the AADT *PairIdentityIdentity*. So the definition of the type *Topology* is based on the type *PairIdentityIdentity*, which is based on the type *Identity*. As a result, in order to unfold *Topology*, one needs to unfold also the other two types. Unfolding of a data type is specified by the name of the type, and the limit that will be considered for the domain. Here is an example of unfolding specification for *Topology* and for its dependencies.

```
Identity : TOTAL;
PairIdentityIdentity : TOTAL;
Topology : 3;
```

The type *Identity* is unfolded to the number of nodes in the network; the type *PairIdentityIdentity* is totally unfolded. That means that all the possible pairs that can be created with the identities of the nodes in the network will be taken into consideration. *Topology* is then unfolded to the desired depth. For example, if the bound is set to 3, AlPiNA will take into consideration all the lists with three pairs that can be constructed with the pairs obtained by unfolding *PairIdentityIdentity* type. This way, we have actually defined all the topologies that a network can have with the given number of nodes, and in which there are three nodes which can communicate directly.

The number of topologies that will be taken into consideration in a non deterministic way through the above unfolding mechanism depends on the number n of nodes in the network, and on the number m of direct connections between them. This value represents the number of combinations of pairs that can be formed with n identities, taken m at a time. As the values for n and m increase, this value is rapidly increasing too. Unfortunately, the topology of the network cannot be abstracted, nor parameterized because of the way message exchange is done in wireless networks. In the case of a broadcast, the nodes which should receive the message can be determined only from the topology. Likewise, in the case of unicast or multicast, the topology is the only information regarding the fact that a node should receive the message or not. In conclusion, the topologies have to be taken into consideration explicitly.

Let us consider an example. If the ad hoc network has three nodes: A, B, and C, it means that for Identity all these three values will be considered. Next, because *PairIdentityIdentity* is totally unfolded, the following values will be considered for it: AB, AC, BA, BC, CA, and CB. As a result, Topology can have the following values:

```
(1) {},
(2) {AB}, {AC}, {BA}, {BC}, {CA}, {CB},
(3) {AB, AC}, {AB, BA}, {AB, BC}, {AB, CA}, {AB, CB},
...,
(4) {AB, BA, BC}, {AB, BA, CA}, {AB, BA, CB},
...
```

With (1) we consider the topology in which none of the nodes have direct wireless connections. With (2) we consider the possible topologies in which only two nodes can communicate directly, the third one being outside the communication range with each of the other two. With (3) we consider the possible topologies in which there are two groups of two nodes which can reach each other. And with (4) we consider all the topologies in which there are three groups of two nodes which can communicate with each other.

If the same value is considered for the topology for a whole protocol run, it means that after considering all these values, the protocol will be verified for all the possible topologies for three nodes. When different values are consider successively in the same protocol run, it means that the protocol is verified over a dynamic topology. So because of the fact that the order in which each of these values is considered is non deterministic, the verification will be made for all the topologies and for all the possible node movements in each of the topologies.

Because it is a list, *Topology* is an infinite data type. So unfolding its entire domain is impossible. But AlPiNA allows the partial unfolding up to a given bound on the number of elements, as we explained above. It is important to state that this second way of defining the topology of the network is particular to AlPiNA and it works thanks to a special characteristic of the verification algorithm called partial net unfolding. Partial net unfolding means that it is not mandatory to unfold all the types, and the user can choose only the type that it needed to be unfolded ([1]).

When the topology is defined as a closed term, AlPiNA will compute the state space for the given algebraic Petri net N, starting from the initial marking. If M_0 is the initial marking, then the state space computed for a given topology can be written as:

$St_N(M_0)$.

When the topology is defined by unfolding, the algebraic Petri net is parameterized by a free variable of type Topology. If $\$tp$ is the name of this variable, then the parameterized algebraic Petri net can be written as:

$N(\$tp)$.

By unfolding, AlPiNA will instantiate the variable $\$tp$ with each of the possible values of the topology, as explained above, thus computing a set of algebraic Petri nets, one for each value:

$N = \cup_{x \in T_{\Sigma,Topology}} N(x)$.

When computing the state space, AlPiNA will actually compute the set of state spaces such that each state space corresponds to a value for the topology. We can write this as follows:

$St_N(M_0) = \cup_{x \in T_{\Sigma,Topology}} St_{N(x)}(M_0)$.

As it will be presented in section 4, the security properties that we have model checked with AlPiNA were expressed through an invariant property. In order to check such a property, AlPiNA starts by computing the state space of the algebraic Petri net provided as input. Then, it checks if the specified property is true for each of the states. If it is, then the property holds for the model. If not, the property does not hold for the model, and a counter-example is provided.

If the topology is defined as a closed term, checking a property for the model implies checking the property for the state space computed for the corresponding APN.

$St_N(M_0) \models invariantproperty$

If the topology is defined by unfolding, checking a property for all models implies checking it for the set of state spaces generated by instantiating the topology variable with all the possible values.

$St_N(M_0) \models invariantproperty \Leftrightarrow$
$\cup_{x \in T_{\Sigma,Topology}} (St_N(x)(M_0) \models invariantproperty)$

So we will check the invariant on all instances; finding a contradiction will mean there is one topology that contradicts the invariant. If the invariant is satisfied on the whole model it means that it is obviously satisfied in each instance.

3.6 Modeling the network

The message exchange in an ad hoc network has special characteristics, because all the nodes act like routers. When a node transmits a message, it is received only by the nodes which have a direct connection to that node. Then, each of the nodes which received the message, processes it according to the routing protocol, and then retransmits it. This process continues until the message gets to the destination. Another aspect that must be taken into consideration is the fact that messages can be of unicast or broadcast type. If a message is unicast, it will be processed only by the node to which it is destined. If a message is broadcast, it should be processed by all the nodes which can receive it directly according to the topology of the network.

The messages transmitted by all the nodes are stored in the place called *Transmitted Packets* (Fig. 3). The network processes the messages from this place and then stores them in the place called *Received Packets* (Fig. 3), from where the nodes can take them for processing and so on.

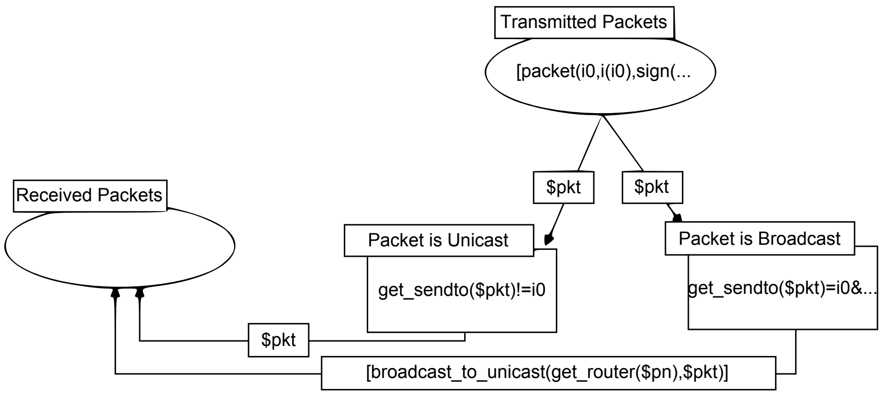

Fig. 3. The model for the ad hoc network operation

In order to have in the High level Petri net model the behavior presented above, we need to model accordingly two elements: the format of the messages exchanged by the nodes and the network itself. Regarding the format of the messages, besides the fields that a message has according to the considered routing protocol, we added two extra fields: a field that stores the identity of the previous node that transmitted it (*prev*), and a field that represents the identity of the node which should receive the message (*next*). If next field contains the value *i0*, then it means that the message is broadcast. Otherwise the message is unicast. The structure of the AADT *Packet* is provided in Fig. 4.

The modeling of the transmission/reception of a message is given in Fig. 3. All the messages transmitted by the nodes are stored in the place called *Transmitted Packets.* From here they are processed in order to provide the behavior explained in the previous paragraph. First we check if the message is unicast or broadcast. If it is unicast, no other processing is required (transition *Packet* is unicast) so the message is placed in the *Received Packets* place from where the destination node can pick it up for processing.

Identity of the node that sent the message	Identity of the node that should process the message/ Broadcast message	Signature(s)			Certificate(s)
		Message type (route discovery request/route discovery response)	Destination node	Nonce	

Fig. 4. The model for the ARAN messages

If the message is broadcast (transition *Packet* is broadcast), we search in the topology for all the identities of the nodes which can receive the message according to it, and we produce the same number of copies for the message, but with the next field filled with the corresponding identity. To verify in the APN if a certain node with identity i can receive a message, we search the variable of type *Topology* if it contains a pair of identities formed by the identity stored in *prev* and by i.

It is worth mentioning that this model of broadcast has an atomicity problem caused by some limitations of the Petri nets. Unfortunately there is no better way of modeling it with the current formalism. The problem is the fact that all the copies of the broadcasted message should reach all the destination nodes at the same time, as if they would be produced in the same transition. This is not possible to model, so, as a result, given the non determinism of the Petri net, other transition could be fired before all the copies reach the destination nodes. This could be solved by an extension of the Petri net, as the one proposed in [20]. The LLAMAS (Language for Advanced Modular Algebraic Systems) model proposed here is based on the old ideas of CO-OPN and it uses synchronization between the transitions in order to provide a better control of the atomicity. By using such synchronization it would be possible to force the correct transmission of a broadcast message by preventing any other transition to fire before the transition that handles the broadcast fires all the possible times. Such a mechanism will also have an impact over the combinatorial explosion by eliminating possibilities that have no meaning in the real ad hoc networks.

3.7 Modeling the adversary

The model that we used for the adversary was the Dolev-Yao model ([21]). In this model it is presumed that the adversary can perform the following operations:

- he can intercept all the messages transmitted in the network (1);
- he can generate new messages based on the knowledge he obtained from the intercepted messages (2);
- he can transmit messages (without modifying them) in the name of any node in the network (3);
- he can prevent a node from receiving a message that was meant for it, with the purpose of sending it another message (4).

Due to the state space explosion problem, we were unable to fully implement this kind of adversary in our model. We have only implemented attack types (1), (3) and (4). To implement attack type (1), the adversary was modeled as having access to all the messages exchanged in the network (places *Transmitted Packets* and *Received Packets* in Fig. 3). Thus he can perform the following actions over the messages: he can drop a message and thus preventing a node to receive it (implementation of attack type (4)) with the purpose of replacing the dropped message with another one, and he can retransmit a message (without modifying it) to another node than the node it was meant for (attack type (3)).

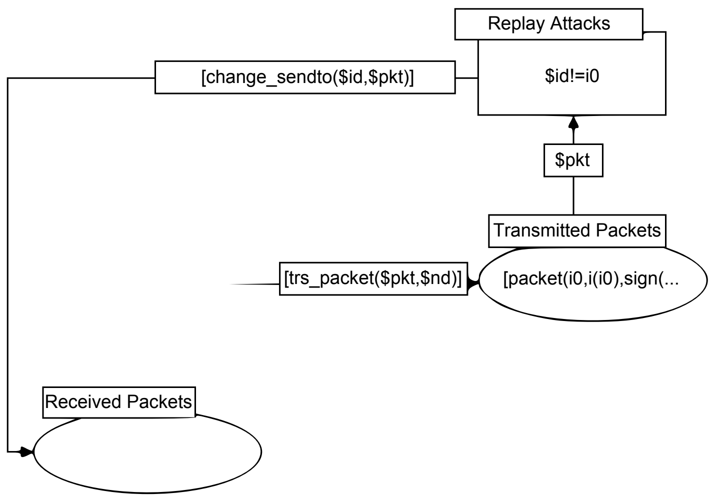

Fig. 5. The model for the adversary

As a consequence, cryptographic security properties like authentication, confidentiality and integrity cannot be checked. Correctness properties can be checked and we will present how in section 4.

3.8 ARAN operation

When modeling ARAN, we have focused on the most important part of the protocol which is the route discovery. As one can see from Fig. 6, the behavior of a node that participates in a route discovery process was modeled with two transitions. The transition *REP Packet at source* corresponds to the fact that the node that initiated the route request receives the response message. The transition *Packet processing* corresponds to all the other processing that a node has to do: broadcast of a route request message by an intermediary node, reception of the route request message by the targeted node, validation of the digital signature(s) from the message, response to a route request message by the destination node, and the unicast of a response to a route discovery message. The actual behavior is implemented by axioms in the AADTs that define the nodes, the messages, the certificates, and the cryptographic operations. The conceptual difference between the two transitions is the presence of the place called *Witness Nodes I*. The purpose of *Witness Nodes I* will be explained in the following paragraph.

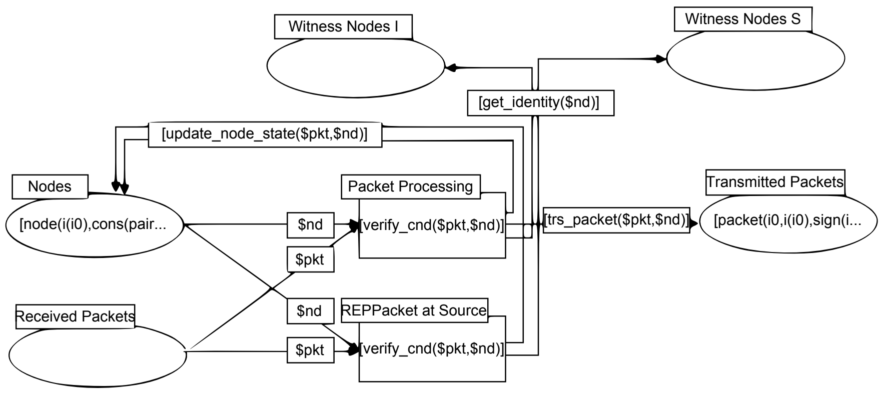

Fig. 6. The model for the node behavior in ARAN

4 Verification of security properties for ARAN

The security objectives of ARAN are to provide authentic and correct routing information for the nodes that issue a route request. Thus, the security properties that have to be verified are authentication of the nodes which participate in the route discovery, and integrity and correctness of the exchanged routing information. ARAN was already modeled and verified using different tools, and we will only cite the latest paper on the subject, [22]. ARAN is successful in assuring authentication and integrity, but an intruder can disturb it by replaying attacks and can propagate incorrect information about the topology of the network. In order to validate our method of modeling using AlPiNA, we wanted to see if we will obtain the same results as the ones already reported by previous research.

The security property that we have verified is correctness of routing information. Authentication and integrity were not considered for reasons explained in section 3 and there are no known attacks against these objectives.

To present what correctness of routing information means, let us consider the topology presented in Fig. 2. If *i(i0)* is the initiator node, and *i^5(i0)* is the destination node, then the expected path between them that should be returned by the protocol is: *i(i0), i^2(i0), i^3(i0), and i^5(i0).* In this case, we say that the protocol provided correct routing information, if and only if for each route discovery request made by node *i(i0)* for node *i^5(i0)*, the protocol will always return the above path. In all the other cases the routing information would not be correct.

In order to verify routing information correctness, we reduced the model of the intruder so that he will only use the possibility of replay attacks. Also, we added to the Petri net the places *Witness Nodes I*, and *Witness Nodes S.* Their role will be presented next. Each time an intermediary node along the routing path from the source node to the destination node processes a message related to the discovery process, its identity is stored in *Witness Nodes I.* The same thing will happen for the destination node too: when it will respond to the route discovery, its identity will be stored in *Witness Nodes I.* In the same manner, when the source node, the node that initiated the route discovery request, will receive the response from the destination node, its identity will be copied to the place called *Witness Node S.*

In the initial marking of the Petri net, the place called *Transmitted Packets* contains a route discovery message from node *i(i0)* for the destination *i^5(i0).* The places *Witness Nodes S* and *Witness Nodes I* are empty. When generating the state space of the model, the place *Witness Nodes S* will eventually contain the identity of the source node *i(i0).* This will mean the protocol run has finished, and the route to the destination was obtained. The identities of the nodes forming the returned route will be in the place *Witness Nodes I.*

To verify the correctness of the routing information, we need to compare the identities of the nodes from *Witness Nodes I* place with the identities of the nodes from the actual path in the considered topology. Using the property specification language available in AlPiNA, we have specified this property in the following way: If the number of nodes in the place *Witness Nodes S* is equal to one it implies that the number of nodes in the place *Witness Nodes I* is equal to the number of nodes in the path from the considered topology. Here is the specification of this property in AlPiNA's property specification language:

```
(card($x in WitnessNodesS) = 1 ) =>
       (card($y in WitnessNodesI) = value );
```

If the property holds when model checking is performed it means the protocol provided correct routing information. Otherwise, the routing information is incorrect and AlPiNA will display a counter-example: content for the place *Witness Nodes I* that contains a different number of nodes. Based on this counter-example we can reconstitute the attack performed by the intruder.

After performing the model checking we have seen that the protocol does not always provide correct routing information, meaning that the intruder was able to mount an attack on it (in concordance with [22]).

Returning to the example we have considered when explaining how the verification is done, when model checking the protocol for this topology, the place

Witness Nodes I, contains *{i^2(i0)}*, or *{i^3(i0)}*, or *{i^5(i0)}*, or *{i^2(i0), i^3(i0), i^5(i0)}*. Only the last value for *Witness Nodes I* corresponds to a correct run of the protocol. The other values represent incorrect routing information that the intruder manages to propagate in the network by replaying attacks. For example, if place *Witness Nodes I* contains *{i^5(i0)}*, it means that the intruder managed to replay the route discovery message sent by *i(i0)* to *i^5(i0)*, and prevented node *i^2(i0)* from receiving it. In this way *i^5(i0)* believes it has a direct connection with A, and responds accordingly. The intruder does the same with the route response message from *i^5(i0)*.

Table 1. Quantitative information

Tool name	Tool's performance for ARAN		
	Number of nodes	*Time (s)*	No of states
AlPiNA	4 (all nodes attacked)	0.95	436
	5 (all nodes attacked)	3.70	4655
	6 (all nodes attacked)	80.82	77239
	7 (6 nodes attacked)	110.95	79131
	8 (5 nodes attacked)	20.22	11637
	9 (5 nodes attacked)	32.92	15500
	10 (5 nodes attacked)	44.06	19363
AVISPA	4	0.05	-
	5	0.07	-

The table above presents quantitative information regarding the verification of routing information correctness, as previously described, in comparison with another model checker called AVISPA, used in [22], where the authors reported the same verification results as we have. The variable of the runs is the number of nodes, besides the adversary, in the topology of the ad hoc network that is taken into consideration. For some of the cases, the tool was unable to compute the state space for all the possible attacks. So we limited the number of nodes which were attacked to some maximum value, which is provided in the table between parentheses, in the same cell as the number of nodes.

AVISPA uses an on-the-fly model checking technique in which attacks are searched for without a prior computation of the whole state space. On the contrary, AlPiNA first computes the entire state space in a symbolic manner, and only then makes the search for attacks. As a consequence, the values provided for AVISPA represent the time of finding the replaying attack for the considered specification, while in the case of AlPiNA, the time column represents the time of computing the entire state space of the considered model. These values cannot be directly compared, but they reveal the fact that AlPiNA is capable of handling the whole state space of the specifications verified with AVISPA, but with the limitation explained above. AlPiNA is capable of handling state spaces of 1-2 millions of states, but in this case, because of the atomicity problem presented at the end of subsection 3.6, starting with 7 nodes, all being attacked, the size of the state spaces goes directly to more millions of states than AlPiNA can handle. This is the reason of using these limitations and also the reason for the fact that the biggest size of the state space in the table is a little less then 80000.

In [22], the authors state they were unable to check the protocol for more than four and five nodes respectively, because of the state space explosion. But using AlPiNA, we managed to model check the protocol for 10 nodes.

5 Conclusions and Future Work

In this paper we have presented the use of algebraic Petri nets for modeling ad hoc networks and for verifying correctness properties for security protocols specially designed for this type of networks, with the use of AlPiNA, a symbolic model checker based on APNs. As far as we know this is the first report of using Petri nets for verifying security properties of the protocols designed for ad hoc networks.

As one can see from the figures we have provided, the Petri net that models the ad hoc network and the security protocol is very simple and clear and has a very small number of places. For example, the model for ARAN has six places. The heavy part of the model is represented by the AADTs that were defined. Thus AlPiNA combines the powerful symbolic model checking with the easy to use APN formalism, providing a good user experience, but also with the ability to master state space explosion.

The limitation of our approach refers to the fact that fabrication attacks were not considered. Fabrication refers to the ability of the intruder to create and send new messages, based on what he previously learned from the network. Our model for the adversary is capable of using the messages he learned from the network, but cannot create new messages. Because it is a symbolic model checker, when an attack is found, AlPiNA cannot provide attack traces. This makes it very difficult to model fabrication attacks, because of the lack of feedback from the tool. But we plan to address this limitation by developing a technique for inversing transitions in an APN, and thus providing attack traces and the necessary feedback.

The model and the verification performed for ARAN secure routing protocol discovered all the attacks that were previously reported for this protocol. This proves the validity of the method, but most importantly, it proves that AlPiNA can be used with success for verifying security protocols.

As future work, we have proposed to perform a quantitative comparison between CPN Tools and AlPiNA in order to see the actual performance improvement brought by the latter. Also we will work on proposing an extension to the current APN model, that will be more adequate to the modeling of distributed protocols, in general, and which, in particular, will be capable of handling broadcast and similar operations in a correct manner. Another future work direction is to modify the modeling of the topology, such that equivalent topologies will be eliminated from the verification, thus reducing the state space and increasing the performance of the model checking.

References

1. Steve Patrick Hostettler, High-level Petri net model checking: the symbolic way, PhD thesis, University of Geneva, 2011.
2. Yongyuth Permpoontanalarp, Panupong Sornkhom, A New Colored Petri Net Methodology for the Security Analysis of Cryptographic Protocols, in The 10th Workshop and Tutorial on Practical Use of Colored Petri Nets and the CPN Tools, Denmark, pp. 81-100. 2009.
3. Didier Buchs, Steve Hostettler, Alexis Marechal, Matteo Risoldi, Alpina: A symbolic model checker, Applications and Theory of Petri Nets, pp. 287-296, 2010.
4. Steve Patrick Hostettler, Alexis Marechal, Alban Linard, Matteo Risoldi, Didier Buchs, High-Level Petri Net Model Checking with AlPiNA, Fundamenta Informaticae, IOS Press, Amsterdam, The Netherlands, vol. 113, no. 3-4, August 2011, ISSN, 0169-2968, pp. 229-264, 2011.
5. AlPiNA tool web page, http://alpina.unige.ch/, the 23 of December 2013.
6. Kimaya Sanzgiri, Bridget Dahill, A Secure Routing Protocol for Ad Hoc Networks, Proceedings of the 10th IEEE International Conference on Network Protocols, pp. 78-87, 2002.
7. L. Khoukhi, S. Cherkaui, Intelligent Solution for Congestion Control in Wireless Ad hoc Networks, in WONS 2006: Third Annual Conference on Wireless On-demand Network Systems and Services, pp. 10-19. 2006.
8. Tzu-Chiang Chiang, Zueh-Min Huang, Multicast Routing Representation in Ad Hoc Networks Using Fuzzy Petri Nets, Proceedings of the 18th International Conference on Advanced Information Networking and Application, vol. 2, pp. 420, 2004.
9. Congzhe Zhang, Mengchu Zhou, A Stochastic Petri Net Approach to Modeling and Analysis of Ad Hoc Network, in Proceedings of the International Conference on Information Technology: Research and Education, pp. 152-156, 2003.
10. Marco Beccuti, Massimiliano De Pierro, Andras Horvath, Adam Horvath, Karoly Farkas, A Mean Field Based Methodology for Modeling Mobility in Ad Hoc Networks, in Vehicular Technology Conference (VTC Spring), 2011, IEEE 73rd, pp. 1-5, 2011.
11. Piyush Prasad, Baltej Singh, Asish Kumar Sahoo, Validation of Routing Protocol for Mobile Ad Hoc Networks using Colored Petri Nets, bachelor thesis, National Institute of Technology, Rourkela, 2009.
12. Chaoyue Xiong, Tadao Murata, Jeffery Tsai, Modeling and Simulation of Routing Protocol for Mobile Ad Hoc Networks using Colored Petri Nets, Proceedings of the Conference on Application and Theory of Petri Nets: Formal Methods in Software Engineering and De-fence Systems, vol. 12, pp. 145-153, 2002.
13. Mohammad Ali Jabraeil Jamali, Tahere Khosravi, Validation of Ad Hoc On-demand Multipath Distance Vector Using Colored Petri Nets, International Conference on Computer and Software Modeling, Singapore, vol. 14, pp. 29-34, 2011.

14. Kristian L. Espensen, Mads K. Kjeldsen, Lars M. Kristensen, Modeling and Initial Validation of the DYMO Routing Protocol for Mobile Ad-Hoc Networks, Applications and Theory of Petri Nets: 29 International Conference, Lecture Notes in Computer Science Volume 5062, pp. 152-170, 2008.
15. Yongyuth Permpoontanalarp, Apichai Changkhanak, Security Analysis of the TMN Protocol by Using Colored Petri Nets: On-the-fly Trace Generation Method and Homomorphic Property, the 8th International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 63-68, 2011.
16. Yang Xu, Modeling and Analysis of Security Protocols Using Colored Petri Nets, Journal of Computers, vol. 6, no. 1, pp. 19-27, 2011.
17. Ulrike Golas, Kathrin Hoffman, Hartmut Ehrig, Alexander Rein, Julia Padberg, Functional Analysis of Algebraic Higher-Order Net Systems with Applications to Mobile Ad-Hoc Networks, Bulletin of the EATCS, no. 101, pp.148-160, June 2010.
18. J. Padberg, H. Ehrig, L. Ribeiro, Formal Modeling and Analysis of flexible Processes in mobile ad-hoc networks, Bulletin of the EATCS, pp. 128-132, 2007.
19. Hartmut Ehrig, Bernd Mahr, Fundamentals of Algebraic Specification 1: Equations and Initial Semantics, Monographs in Theoretical Computer Science, An EATCS Series, Springer, 1985.
20. Alexis Ayar Marechal Marin, Unifying the syntax and semantics of modular extensions of Petri nets, PhD thesis, University of Geneva, 2013.
21. Danny Dolev, Andrew Yao, On the Security of Public Key Protocols, IEEE Transactions on Information Theory, vol. IT-29, nr.2, pp. 198–208, 1983.
22. Davide Benetti, Massimo Merro, Luca Vigano, Model Checking Ad Hoc Network Routing Protocols: ARAN vs. endairA, The 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM), pp. 191-202, 2010.