DAV International Journal of Science

FORMALIZATION AND VERIFICATION OF A WIRELESS NETWORK PROTOCOL BY USING SYMBOLIC MODEL CHECKING

Alireza Souri [1]* and Monire Norouzi[2]
[1]Young Researchers and Elite Club, Tabriz Branch, Islamic Azad University, Tabriz, Iran
[2]Young Researchers and Elite Club, Shabestar Branch, Islamic Azad University, Shabestar, Iran
*(E-mail: alirezasouri.research@gmail.com)

ABSTRACT
Formal verification is one of the methods for evaluating the computer systems. In this paper we used SMV model checker to evaluate the performance of DSR protocol in wireless networks. Since the modeling of whole DSR network in SMV is not possible, we used a Case Study example. In this example, node failure and link break conditions are investigated. By using SMV model checker, we have been able to detect routing problems in these situations and guarantee the safe arriving of data packet to destination by providing an algorithm. Implementation results show that SMV model checker is one of the tools for verifies the wireless routing protocols.

KEY WORDS: Formal verification; wireless networks; model checking; DSR protocol; SMV; computational tree logics.

INTRODUCTION

Ad hoc networks are based on specific routing protocols. Unlike in wired networks that have some special nodes to perform as routers, in the ad hoc networks every node must acts as a router to being message for the others. Because no infrastructure exists, the nodes that want to communicate but are not in one another's coverage area must rely on the intermediate nodes to forward their messages. This means that when a node wants to send some data to another one, it first has to determine the path in the network that the data must follow in order to reach the destination. This is called the route discovery phase. After that follows the data is forwarding to phase, in which the actual message is sent to the destination by using the path previously determined (Pura, Patriciu et al. 2010).

Protocol verification is the process of examining the protocol specifications for the presence of various errors that could lead to improper operation. Formal verification is better than testing by implementation (Holzmann 1997)because:
Here only the existence of errors might be Detectable.

- It is less costly to fix errors.
- It is independent of implementation.
- It is fully automated.

Some previous works have been done on the verification of wired routing protocols and even wireless routing protocols (Bhargavan, Obradovic et al. 2002).for example, using SPIN (Holzmann 2003) and PROMELA (Ribeiro, Fernandes et al. 2005) to verify ATM Routing Protocol PNNI. However, these approaches only handle very small number of nodes (typically two or three nodes) , extremely simple and fixed network topology (Royer and Perkins 1999).

We discuss that although their models can provide an accurate representation of a single node's behavior, but they do not provide satisfactory solution to state of exclusive problem, therefore cannot be extended to handle even slightly more complicated network topology. Also, fixed network topology is surely an unreasonable assumption for wireless protocol since in real environment nodes tend to be very dynamic and can move around without any restriction. This property suggests a dynamic network topology (Bugliesi, Gallina et al. 2014).

As an important step towards this goal, in this paper we propose a formal method (Lv 2012) to decompose the DSR protocol into representative categories and use a step by step approach to verify the correctness of routing. We also provide a simple example to handle the modeling of link breaks, node failure ,error messages and improper operations of DSR by SMV.

The systematic nature of our method and its well-founded semantics ensure that one can have much more confidence in a correctness proof obtained with our method than in a "proof" based on informal arguments. In addition, compared to previous approaches that attempted to formalize the verification process of ad-hoc network routing protocols, the novelty

of our approach is that it can be easily configured to handle more complex network and verify more complicated properties in our case study.

The content of this paper is organized as follows: Section 2 gives an overview to Formal Verification. Section 3 gives a brief introduction of dynamic source routing algorithm and describes some terminology which we will use later. In section 4, we give some details about our approach to break the problem set and how to model them with SMV model checker. The bug which we found during the verification and some thoughts about the protocol specification is described in this section. Section 5 presents conclusion and future works.

FORMAL VERIFICATION

Formal verification of a computing system entails a mathematical proof which shows the system satisfies its desired property or specification. To do this, we must use some mathematical structure to model the system of interest and derive its desired properties as theorems about the structure. The principal distinction between the different formal verification approaches some stems from the choice of the mathematical formalism used in the reasoning process (Dong, Peng et al. 2010). By using a formal verification tool, all possible execution of the algorithm design are mathematically analyzed without any need to develop simulation input stimulus or test (Souri and Jafari Navimipour 2014).

SMV model checker

One of the important discussions for verifying and testing software systems or model-based reactions is System of Environment Compatibility with set rules of the system which is expected. Symbolic Model Verifier (SMV), a powerful tool for presentation of analyzing and testing is a system behavior.

According to the SMV inputs that software receives, it can analyses the limited states of a system by using the OBDD (Ordered Binary Decision Diagram) based on the Symbolic Model Checker. In other words, SMV provides the context in which we can implement a system or model. Then we translate some sets of expected behavior of the model or systems into the special logic called CTL and verify them in the implemented model (Iosif 2000). The model checker gives us some results that can be true or false. We can see false of the results as a table named Counter example.

Computational Tree Logic

Computation tree logic (CTL) is branched time logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realized. It is used in formal verification of software or hardware artifacts, typically by software applications known as model checkers that determine if a given artifact possesses safety or live properties. It is in a class of temporal logics that includes linear temporal logic. Temporal properties are expressed in CTL, which is an extension of CTL. The Syntax of CTL is

$$S :: = p \mid \neg p \mid S \wedge S \mid S \vee S \mid Ap \mid Ep \mid Gp \mid Fp$$
$$P :: = \theta \mid \neg\theta \mid S \mid P \wedge P \mid P \vee P \mid Xp \mid P\ U\ P$$

Where S is state formula, P is path formula. p is atomic proposition, and θ is atomic action proposition. A is a universal quantifier which means that the formula is true in all paths starting from the current state. E is an existential quantifier which means that there is a path following the current state, such that the formula is true. G is a path universal quantifier which means that the formula is true for all the states along the path from the current state. F is a path existential quantifier which means the formula is true in some state in the path from the current state. X is a path quantifier which means the formula is true in the next state in the path from the current state. U is union of between two variables. G, F and X are always used together with A and E (Pura, Patriciu et al. 2010).

Wireless networks

DSR

The Dynamic Source Routing protocol (DSR) is a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes. DSR allows the network to be completely self-organized and self-configured, without any need of existing network infrastructure or administration. The protocol is composed of the

two main mechanisms of "Route Discovery" and "Route Maintenance", which work together to allow nodes to discover and maintain routes to arbitrary destinations in the ad hoc network.

To send a packet to another host, the sender constructs a source route in the packet's header, by giving the address of each host in the network through which the packet should be forwarded in order to reach the destination host. Then, sender transmits the packet over its wireless network interface to the first hop which has been identified in the source route. When a host receives a packet, if this host is not the final destination of the packet, it simply transmits the packet to the next hop which has been identified in the source route in the packet's header. When the packet reaches its final destination, the packet is delivered to the network layer software on that host. Each mobile host is participating in the ad-hoc network and maintains a route cache which source uses. When one host sends a packet to another host, the sender first checks its route cache for a source route to the destination. If a route is found, the sender uses this route to transmit the packet. If no route is found, the sender may attempts to discover one by using the route discovery protocol (Ghassemi, Fokkink et al. 2011).

DSR verification

For DSR routing protocol various improvements has been presented. For example, Route cache, Piggybacking and Error handling (Buttyan and Ta Vinh 2010) to increase efficiency and efficient routing have been proposed. Most of these improvements are for solving routing failure problem and non-delivery data packet properly.

Broken links and node failures are problems of routing wireless networks. If the link break or node failure occurs in DSR, data delivery will be difficult. Therefore, verification operation in DSR routing protocol can be a very convenient way to check and find routing error.

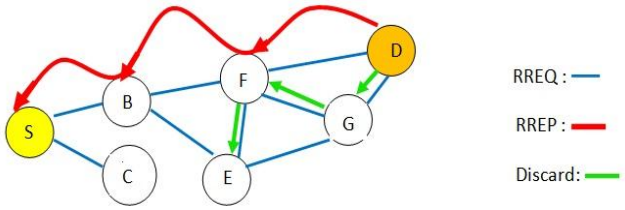

Figure 1. Example of DSR

Before discussing the proposed model, it is helpful to describe the specification of the sample DSR protocol. There are 7 nodes in our case study (fig. 1). In order to analysis the problem, we assume that variables are the nodes.

Type definition: nodes {S, B, C, F, E, G, D};

When a node sends a data packet to intended node, if the receiver node exits from the range of the sender -when it sends data- the sender will release an error message towards the source node. We assume that the receiver node (when sends data) remains in the coverage range of the sender node, if the receiver node stops working any way or faces with the node failure mode, the sender node must wait a threshold and after receiving response, it will begin to discover a new path.

To create and produce this threshold, we should find average time interval between two consecutive nodes in the path. In order to this action, RREQ and RREP messages can be used in route discovery process.

When B node receives RREQ message from S node, B node registers RREQ received time in its cache for the S node. This process will continue until the RREQ reaches to D node. Now, D node must send the RREP message by the reverse address in its header to the source, to confirm the route discovery. D node (simultaneous with sending RREP), sends recorded time of the previous node to F node. Upon receiving RREP, F node stores the average interval time into the D in its cache. With this, when the RREP reaches to source node namely S, all nodes in the current route, approximately have a time of sending packet to its coverage node. The estimated distance is displayed with t_i.

However, after discovering the path, sender must send data packet. S node sends the data to the B node and waits $2t_s$. If before this time, Ack or Error messages do not reach to S node, B node must be in trouble or node failure mode is encountered. So, S node begins to discover new path quickly.

DSR to SMV

To implement the system within the SMV, we should define the following terms. Each node can place values or variables in itself in order to change the mode of system with assigning each of the following objects to itself:

Objects: {send, received , in_cache , cover , sendback } ;

Connectors in the system are defined as follows:
Connector: { Req_msg , Rep_msg , Data_msg , Ack_msg , Error_msg } ;
Main reason of system current and changing conditions is duty of connectors. Each node can create one of the above modes in the system by connecting to an object.
In the average interval time section, we have following variables:
Time : { T_wait , TS , TB, TC ,TF ,TE ,TG , TD } ;
T_wait is a time that one node waits after sending the Data_msg. TS, TB, TC, TF, TE, TG, TD are the times that are stored in cache of each node and are expressed the average time between two sequential nodes.

However, we present the main body of the system in SMV. First, we run the route discovery operation. During route discovery, the interval time between each node are produced by Req_msg, Rep_msg. For example, after sending Req_msg message by S node, B node checks its In_cache section. If S node was in coverage of B node, the B.In_cache = S.cover condition is true and Req_msg message have been arrived to the B node and the time between B and S nodes is stored in T_S. Then, the content of T_S puts in In_cache section of B node. This routine will be continued until arriving Req_msg message to destination node. After that, D node is sending Rep_msg message with T_S to F node at the same discovered route. When F node received Rep_msg message the time between F and D nodes that is named T_F is stored in in_cache section of F node. This routine will be continued up to receiving Rep_msg message to the source node.

```
MODULE   Route Discovery
VAR
Req_msg : Boolean;
Rep_msg : Boolean;
T : { TS , TB, TC  ,TF  ,TE  ,TG  , TD } ;
ASSIGN
init  := S ;
S.send = Req_msg ;
if ( B.In_cache = S.cover)
   B.recieved = Req_msg ;
B.In_cache = TS ;
B.send = Req_msg ;
if ( F.In_cache = B.cover)
    F.recieved = Req_msg ;
F.In_cache = TB ;
F.send = Req_msg ;
if ( D.In_cache = F.cover)
    D.recieved = Req_msg ;
D.In_cache = TF  ;
D.send = Rep_msg & TF  ;
F.Recieved  = Rep_msg ;
F.In_cache = TF ;
F.send = Rep_msg & TB  ;
B.recieved = Rep_msg ;
B.In_cache = TB ;
B.send = Rep_msg & TS  ;
S.recieved = Rep_msg ;
S.In_cache = TS  ;
1 : S;
esac ;
```

After Route Discovery operation, S node starts sending data in discovered path:

```
MODULE Send Data
VAR
Data_msg : Boolean;
Ack_msg  : Boolean ;
In_Cache : Boolean;
```

```
ASSIGN
In_Cache := false ;
init(state) := S;
next(state) := case
state = S & Data_msg: B ;
if ( B.In_Cache = true)  -> (state = B & Ack_msg: S) ;
state  = B & Data_msg: F ;
if ( F.In_Cache == true)
state = F & Ack_msg: B ;
state = F & Data_msg: D ;
 if ( D.In_Cache == true)
state = D & Ack_msg: F ;
1 : state ;
esac ;
```

Now, we show the implementation of link break and node failure modes. In this section S node starts to send Data_msg to destination node in the discovered path. S node checks its In_cache section, if B node was not in coverage of S node, S.sendback = Error_msg command is execute. Otherwise, S node sends Data to B node. After that, S node waits as period of T_S. If the period of T_S passed and S node did not receive Ack_msg message, S.sendback = Error_msg command is execute. Otherwise Ack_msg message has been arrived to S node and mentioned routine will be done for the next node until Data arrives to the destination node correctly and safety.

```
MODULE Node Failure or Link Break

VAR
Data_msg : Boolean;
Ack_msg  : Boolean ;
Error_msg  : Boolean ;
T_out : Boolean ; //  Node waiting time for ACK Message
T : { TS , TB, TC  ,TF  ,TE  ,TG  , TD } ;
ASSIGN
init := S ;
if ( S.In_cache != B.cover ) -> S.sendback = Error_msg ;
else   ->  {  S.send = Data_msg ;
while ( T_out != TS )
   waite ;//
if ( S.recieved  !=Ack_msg )  -> S.sendback = Error_msg ;
            } ;
if ( B.In_cache != F.cover ) -> A.sendback = Error_msg ;
else   ->  {  S.send = Data_msg ;
while ( T_out != TB )
   waite ;//
if ( B.recieved  !=Ack_msg )  -> B.sendback = Error_msg ;
             } ;
if ( F.In_cache != D.cover ) -> F.sendback = Error_msg ;
else   ->  {  F.send = Data_msg ;
while ( T_out != TF )
   waite ;//
if ( F.recieved  !=Ack_msg )  -> F.sendback = Error_msg ;
              } ;
if (S.recieved = Error_msg )  - >
// **Return to Route Discovery
esac ;
```

If state of node failure or link break occurred, eventually Error_msg reaches the source node and begins to discover new path again from the area that Error_msg have been sent. We also know this theorem that when a node realizes that there was a problem for the receiver node, by Error_msg, it gives order to other nodes that remove corrupted node within its coverage range.

Finally, we create the main part of implementation. In this section, there are set of rules that should be verified:

```
MODULE main
VAR
state : { S,B,C,F,E,G,D };
Connector: { Req_msg , Rep_msg , Data_msg , Ack_msg ,  Error_msg } ;
T : { T_out , TS , TB, TC  ,TF  ,TE  ,TG  , TD } ;
Object: { send , received , in_cache , cover , sendback } ;
SPEC EG ( (T_out > TS | TB | TC  | TF  | TE  | TG  | TD )& ( ! Ack_msg )  ->   EX (Error_msg ) )
SPEC  EG ( ( In_cache = cover ) ->  EX (Data_msg) )
SPEC  AG ((In_cache != cover)  ->  EX ( Error_msg ) )
SPEC  AG  ( (Error_msg )  & EF(Data_msg  -> Ack_msg))
MODULE Route Discovery
MODULE  Send Data
MODULE  Node Failure or Link Break
```

RESULTS AND DISCUSSION

In practice, modeling DSR protocol is very large and wide. The protocol is used on sensitive and important positions, because the resources and other nodes are moving in their environment. Considering the above factors, in order to test and verify this model, it took in the smaller size, so we can verify in the SMV Model Checker by CTL Formula features and its properties. In general, for N nodes in DSR routing protocol, there are (N-1) number of paths to other nodes in the standard conditions. According to the following equation, all paths and data sending operations are calculated for N nodes:

$$(n-1)_i(Con)(Obj) + (n-1)_i(Con)(Obj) + \cdots + nNODE \ldots$$

$$+(n-1)_i(Con)(Obj) = \sum_{i=1}^{n} (n-1)_i(Con)(Obj)$$

n: number of nodes
Con: number of Connectors
Obj: number of Objects

After verification of the given model with 7 nodes, SMV model checker has presented results for the existing rules in the system which is specified in the below table:

VERIFICATION RESULTS BY SMV

Property	Result	User Time	System Time
EG ((T_out > TS | TB | TC | TF | TE | TG | TD) & (! Ack_msg) -> EX (Error_msg))	True	0.9891007	5.9346
EG ((In_cache = cover) -> EX (Data_msg))	True	0.9891007	5.9346
AG ((In_cache != cover) -> EX (Error_msg))	True	0.9891007	5.9346
AG ((Error_msg) & EF(Data_msg -> Ack_msg))	False	0.3245021	1.03356

With respect to the above results, the existence rules in the system have been verified with SMV model checker. With verifying these rules and specifications we can evaluate the correctness of existence algorithms in the system and conclude that the existing algorithms in the implementation are caused to reduce the any problem and factors such as starvation and deadlock for network and nodes when node failure and link break occurred. More importantly, in wireless networks, safe arriving of data packets to destination has much influence on system performance. With these implementation and results, we can largely improve safe arriving data packet to destination.

CONCLUSION AND FUTURE WORK

By using SMV software, we were able to implement the operation of DSR routing protocol on a case study in wireless networks. Then we present the timing algorithms to improve network performance and prevent node failure and link breaks modes. To test and verify the algorithm, we used the formulas and rules of the CTL. Finally, obtaining results

from SMV, we saw that the structures presented in the DSR protocol model has the correct answer and has improved the performance of this protocol. According to the obtained results and our experiences, we believe that the SMV model checker can be one of the useful tools to verify the systems and wireless networks. Also the formal verification is one of the most effective techniques for testing, evaluating and obtaining the desired results in wireless networks.
In the future, we will try to implement the larger size of wireless networks in particular security discussion. Also we can test and evaluate the network with other algorithms by more number of states which will increases the security and decreases the system error percentage.

REFERENCES

Bhargavan K., et *al.* (2002). "Formal verification of standards for distance vector routing protocols." *J. ACM.* 49(4): 538-576.
Bugliesi M., *et al.* (2014). "Behavioural equivalences and interference metrics for mobile ad-hoc networks." *Performance Evaluation.* 73(0): 41-72.
Buttyan L. and T. Ta Vinh (2010). Formal verification of secure ad-hoc network routing protocols using deductive model-checking. Wireless and Mobile Networking Conference (WMNC), 2010 Third Joint IFIP.
Dong J., *et al.* (2010). "Automated verification of security pattern compositions." Inf. Softw. Technol. 52(3): 274-295.
Ghassemi F., *et al.* (2011). "Verification of mobile ad hoc networks: An algebraic approach." *Theoretical Comp. Sci.* 412(28): 3262-3282.
Holzmann G. (2003). Spin model checker, the: primer and reference manual, Addison-Wesley Professional.
Holzmann G. J. (1997). "The model checker SPIN." Software Engineering, IEEE Transactions on 23(5): 279-295.
Iosif R. (2000). Formal verification applied to Java concurrent software. *Proc. 22nd Int. Con. Software Engineering. Limerick, Ireland, ACM:* 707-709.
Lv J. (2012). Scalable formal verification of finite field arithmetic circuits using computer algebra techniques, University of Utah: 121.
Pura M.-L., *et al.* (2010). Formal verification of secure ad hoc routing protocols using AVISPA: ARAN case study. Proc. 4th Con. *Euro. Computing Conf. Bucharest, Romania, World Scientific and Engineering Academy and Society* (WSEAS): 200-206.
Ribeiro O. R., *et al.* (2005). Model checking embedded systems with PROMELA. Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE Int. Con. Workshops.
Royer E. M. and C. E. Perkins (1999). Multicast operation of the ad-hoc on-demand distance vector routing protocol. *Proc. 5th Annual ACM/IEEE Int. Con. Mobile Computing Networking. Seattle, Washington, USA, ACM:* 207-218.
Souri A. and N. Jafari Navimipour (2014). "Behavioral modeling and formal verification of a resource discovery approach in Grid computing." *Expert Systems with Applications*. 41(8): 3831-3849.