SensorChecker: Reachability Verification in Mission-Oriented Sensor Networks

Ehab Al-Shaer, Qi Duan, Saeed Al-Haj
University of North Carolina
at Charlotte
{ealshaer, qduan, salhaj}@uncc.edu

Moustafa Youssef
Egypt-Japan University of Science and
Technology
moustafa.youssef@ejust.edu.eg

ABSTRACT

This paper presents novel techniques to verify the global reachability in mission-oriented wireless sensor networks (Mission-Oriented WSN). The global reachability verification considers configurations such as forwarding information and awake/dormant schedule as generated by WSN protocols and algorithms. Our contribution is two-fold. First, we create a scalable model that represents the end-to-end reachability of WSN based on node configuration using Binary Decision Diagrams (BDDs) and Symbolic Model Checking, and then define generic reachability properties using Computational Tree Logic (CTL). Second, we encode the Mission-Oriented WSN topological information using Boolean functions to verify constraint-based reachability properties for WSN, and show soundness and completeness.

We implement this in a tool called SensorChecker. The scalability and performance of SensorChecker is validated with very large WSN networks (10s of thousand of nodes) and wake-up scheduling parameters. To the best of our knowledge, this is the first formal approach for verifying large-scale WSN network configuration.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*

Keywords

Reachability; Verification; Computational Tree Logic; Model Checking

1. INTRODUCTION

Mission-oriented wireless sensor networks usually consist of multiple static/mobile sensors to provide monitoring and data aggregation tasks. Moreover, the new development in hardware miniaturization and wireless communication made the applications that involve large numbers of sensors possible. The wide deployment of WSNs has resulted in developing different protocols to support different tasks. Wireless or wired nodes are configured by WSN protocols to collectively detect and forward sensed data accurately and timely to a designated gateway. Thus, the soundness and completeness of reachability and coverage properties are key requirements for any WSN. However, misconfigurations due to incorrect protocol specification or implementation might cause global malfunctioning in the reachability functions of WSNs.

It is challenging to meet the reachability requirement necessary for achieving the mission of the WSNs. First, many nodes such as wireless sensors have only limited energy and computational capabilities, which require the nodes to frequently switch from active to dormant mode to prolong the network lifetime. Wired devices in WSNs may also have the active/dormant status conversion to save energy. Second, multipath routing is common in WSNs because the routes in WSNs are more unreliable than those in wired networks. To verify reachability in multipath routing is not a trivial task. Third, the additional requirements for reachability in WSNs, such as cost constraints in reachability, make the verification more intractable. These characteristics of WSNs make the design, configuration, and validation of WSNs very challenging and error-prone tasks. Moreover, the existence of a large variety of the protocols and algorithms for configuring WSNs exacerbates the debugging and testing problems in this domain.

In this paper, we present a novel approach to model and to verify the global reachability behavior of Mission-Oriented WSNs based on topology and configuration information. The presented model can be used to validate the reachability and coverage requirements or identify counter examples useful for debugging WSN routing protocols and scheduling algorithms. In our approach, we model a WSN as a state machine encoded using Binary Decision Diagrams (BDD) [2] such that states represent data streams at any location and any time slot, and transitions represent the data transformation according to sensors forwarding actions and status. Our main contribution in this work is creating an efficient abstraction and encoding of WSN behavior to obtain an accurate, scalable, and flexible model for verifying reachability properties of the WSNs regardless of protocol or algorithm specifics. In addition, our system can also verify constraint-based reachability properties for reliability and quality of service requirements. The compact representation of BDDs, the efficient search of symbolic model checking [3] and rich-

MiSeNet'13, October 4, 2013, Miami, Florida, USA.

http://dx.doi.org/10.1145/2509338.2509344.

ness of temporal logic such as Computational Tree Logic (CTL) [4] can facilitate reachability and coverage verification.

SensorChecker is an off-line tool and it does not interfere with the operations of deployed Mission-Oriented WSNs. It basically reads the topological and configuration information of the WSN as an input, and it allows users to generate any CTL-based query to verify the future reachability behavior of the system. If the query is unsatisfied, the system produces a Boolean expression as a counter example for debugging and root cause analysis. The presented model takes in consideration the dynamic nature of a WSN; the model can be incrementally (*i.e,* partially) updated to reflect the new behavior. The model will be completely reconstructed only if the changes are significant.

In previous work, number of techniques have been proposed to verify sensor network protocol specifications [6,7,9]. However, unlike SensorChecker, these techniques are limited to specific protocols and properties. Other approaches were proposed to verify network security configuration such as ConfigChecker [1]. However, the ConfigChecker encoding is utilized mainly for IP networks and it is not applicable to wireless sensor networks due to dynamic changes in node activity based on awake/dormant schedule.

2. SYSTEM ASSUMPTIONS FOR MISSION-ORIENTED WSNS

We assume a typical Mission-Oriented WSN where sensor nodes are deployed to monitor an area of interest along with base stations to collect the data. We do not assume a specific deployment distribution. Base stations are not energy constrained. Nodes' locations are assumed to be known so that the sensed data can be tagged with location information.

In our model, we assumed that a WSN node will choose the first match among all awake next hops to forward a packet. If there is no match (i.e. all next hops are dormant), then the sensor node must wait for a period of time until itself and one of its next hops are awake. Our model is general enough to include different classes of protocols.

Before the reachability analysis can be done, the topology and forward scheduling information of the network must be known. The Base Station can collect the position information of the nodes and use it to build the network topology. This can be done in a separate bootstrapping phase, or it can be piggybacked on the data packets sent from WSN nodes to the Base Station. For the packet forwarding scheduling information, this can be achieved by different ways depending on the routing protocol. For a centralized routing protocol, the Base Station already has this information. For a distributed routing protocol, a collection mechanism similar to the one used in the topology construction can be used.

The reachability analysis can then be done offline and will not interfere with WSN operations. Our model is not restricted to specific protocols, most of the protocols can work with our model. Any protocol can be abstracted by time schedule and routing information for each node. SensorChecker will use timing and routing information gathered from each node to check if a specific reachability requirement is enforced or not at any period of time. Based on the results, protocol bugs or node misconfigurations will be reported.

3. BASIC MODEL OF WSN CONFIGURATION TRANSITIONS

3.1 Time Slot Based State Representation

SensorChecker models the WSN as a finite state machine such that the state transitions represent information flow from one node to another node. After receiving a packet, a WSN node may take a certain action according to the source address, destination address, and the current time slot number. We can express this as the following characteristic function:

$$\sigma : \mathbf{Src} \times \mathbf{Dest} \times \mathbf{Loc} \times \mathbf{Time} \rightarrow \{\mathbf{true, false}\}$$

Src: the ID of the source node.

Dest: the ID of the destination node.

Loc: the ID of the node currently processing the packet.

Time: the index of a time slot.

The function σ encodes the state of the network. A state is evaluated to *true* whenever the input parameters correspond to a packet that is in the network, and *false* otherwise. For example, if the network contains 5 different packets, then exactly five assignments to the parameters of the function σ will result in true.

A node may enqueue the packet if no appropriate next hop is awake, or may forward it to some awake neighbors according to the packet destination (usually the destination is the base station). The node may also add some additional information for future security check, or may change the source and destination information in some protocols. For simplicity, we defined two modes of operation: awake and dormant. When the node is awake, it will do data sensing, packet receiving, and forwarding. When a node is in dormant mode, it will not do any receiving and forwarding. We encode the dormant status as 0 and awake status as 1.

3.2 Building State Transitions

State transitions in the model correspond to packets moving between nodes in the WSN. Our goal here is to build a single BDD for the whole system. BDDs can provide an efficient data structure for the representation of sets and relations, and are extensively used in logic synthesis and formal verification in recent years. Algorithm 1 shows how to build the transition BDD for an arbitrary WSN.

Note that in the algorithm, $MaxSlot$ is the maximum time slot number, and $awake(x, j)$ means sensor x is awake at time slot j. $\mathbf{Dest}'$, $\mathbf{Loc}'$, $\mathbf{Time}'$ are the next step variables in the transition. The overall transition BDD is the disjunction of all possible transitions for all sensor nodes in all time slots. We assume that forwarding a packet takes only one time slot. Any node can receive packets at a time slot when it is awake. After receiving a packet, the receiving node needs to find the first time slot that both itself and at least one hop that leads to the destination will be awake, and forward the packet to the first matched next hop at that time slot.

As an example, we used two bits ID to represent node address, and two bits to describe the time slot. The following Boolean variables are used:

Algorithm 1: Time Slot Based Transition Generation Algorithm

$T = bddfalse$;
$i = 0$;
while $i < MaxSlot$ **do**
  for *every sensor node* x **do**
    if *node* x *is awake in time slot* i **then**
      for *every destination* d *in the forwarding table* **do**
        find the next hop k and smallest j such that $awake(x, j)$ and $awake(k, j)$ and $j \geq i$;
        $T = T \vee ((\mathbf{Src} = s \wedge \mathbf{Dest} = d \wedge \mathbf{Loc} = x \wedge \mathbf{Time} = i) \wedge (\mathbf{Src}' = s \wedge \mathbf{Dest}' = d \wedge \mathbf{Loc}' = k \wedge \mathbf{Time}' = j + 1))$;
      end
    end
  end
  $i = i + 1$;
end

s_1, d_1, l_1: The high order bit of the source address, destination address, and the location ID respectively.

s_0, d_0, l_0: The low order bit of the source address, destination address, and the location ID respectively.

t_1, t_0: The two bits of the current time slot

Variables with a prime sign are the next state variables, for example, s'_0, d'_0, l'_0 are the low order bit of the source address, destination address, and the location ID of the next state in the transition. All variables are Boolean variables, the Boolean representation is encoded in the transition function as follows: s_0 means $s_0 = 1$ and $\overline{s_0}$ means $s_0 = 0$.

As an example, suppose a node with ID 1 is awake in time slot 2. The node only processes packets with source ID 0 and destination ID 2 and discards all other packets. According to the routing rule, node 1 will forward the packet to node 3, which is also awake in slot 2. We assumed that the current time slot is 2. Then the state transition can be described by the following Boolean formula:

$$(\overline{s_1} \wedge \overline{s_0}) \wedge (d_1 \wedge \overline{d_0}) \wedge (\overline{l_1} \wedge l_0) \wedge (t_1 \wedge \overline{t_0})$$

The previous formula means: source ID is 0, destination ID is 2, current node ID is 1, and the current time slot is 2.

Also, in the new state, the packet will look identical, except for the new location and the new time slot, so the restriction for the new state is

$$\bigwedge_{i \in \{0,1\}} (s'_i \Leftrightarrow s_i) \bigwedge_{i \in \{0,1\}} (d'_i \Leftrightarrow d_i) \wedge (l'_1 \wedge l'_0) \wedge (t'_1 \wedge t'_0)$$

The transition relation for this node (with ID 1) at time slot 2 is the conjunction of the above two relations.

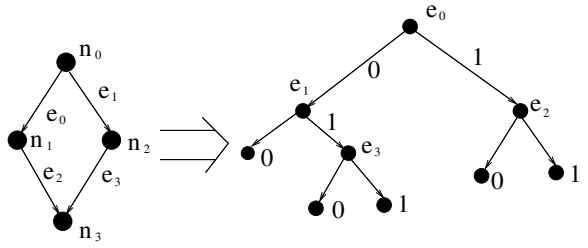

Figure 1: BDD representation of paths in a graph

4. MODELING WSN REACHABILITY WITH COST AND RELIABILITY CONSTRAINTS

In this section, we present a new model that can deal with constraint based queries such as "reachability between source and destination with paths that have cost less than 10". Here the path costs in WSNs are typically the transmission power cost along the path, number of hops, or some quality of service metrics.

Given the WSN topology, we can construct a BDD to represent satisfiable paths from a certain source to a certain destination. For example, Fig. 1 shows the BDD representation for a graph that connects n_0 to n_3. The left hand side of the figure is the original graph representation of the network, and the right hand side of the same figure is the constructed BDD to represent the paths from node n_0 to n_3 with length that is no more than 2 hops. Every node in the BDD represents an edge in the network. The left branch (denoted as 0) of every edge means not choosing the edge for a satisfiable path, the right branch (denoted as 1) means choosing the edge for a satisfiable path. Any leaf node marked as 1 means a satisfiable path. In this figure, there are two satisfiable paths from n_0 to n_3; the first one is by choosing e_0 and e_2 and the second one is by not choosing e_0 and choosing e_1 and e_3. In Fig. 1, if node n_0 is connected to more than two nodes as shown in the figure, then more edges are needed to encode the network topology.

The algorithm to generate the BDD is shown in Algorithm 2, which is an extension of the fault tree analysis algorithms in [8].

In Algorithm 2, a BDD is constructed to represent all the possible simple paths (i.e, no cycles) that are from a specified source and destination pair and satisfy the cost constraint. Here, $bdd(e)$ is the BDD variable representing an edge. Edge variables of any satisfiable assignment of the constructed BDD are the edges in a satisfiable path.

We can easily extend the algorithm to handle paths with multiple constraints as long as the constraints can be computed additively or multiplicatively along the nodes or edges of a simple path.

5. WSN CONFIGURATION VERIFICATION USING SYMBOLIC MODEL CHECKING

To facilitate configuration verification in Wireless Sensor Networks, we use Computational Tree Logic (CTL) [4] for query construction. CTL is branching-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 real-

Algorithm 2: BDD generation for reachability with cost constraints

```
bddgen(start_node, total_cost)
if cost of start_node > total_cost then
  | return bdd_false;
end
T_bdd = bdd_false;
S = Φ;
add start_node to S;
for every edge e of start_node do
  | i = the other end node of the edge ;
  | if i is the destination then
  |   | path_bdd = bdd(e)
  | end
  | else if i is already in the this path then
  |   | continue;
  | end
  | else
  |   | path_bdd =
  |   | bdd_gen(i, total_cost − cost(start_node)) ∧ bdd(e)
  | end
  | T_bdd = T_bdd ∨ path_bdd
end
remove start_node from S;
return T_bdd;
```

ized. CTL operators **AF** and **EF** will be used in reachability queries.

In this paper, we focus on queries related to reachability. The following discussion describes how to construct a query BDD related to reachability.

For the time slot based transition model, the query to verify basic reachability between a sensor node i and BS for all possible forwarding choices at a specific time slot can be:

$$(src = i \wedge dest = BS \wedge loc(i)) \rightarrow$$
$$\mathbf{AF}(src = i \wedge dest = BS \wedge loc = BS \wedge time = j)$$

Here **AF** means that the statement will be true for all possible outcomes. We define this query to be $R_{AF}(i, BS, j)$.

The query to verify basic reachability between a sensor node i and BS for some possible forwarding choices at a specific time slot can be:

$$(src = i \wedge dest = BS \wedge loc(i)) \rightarrow$$
$$\mathbf{EF}(src = i \wedge dest = BS \wedge loc = BS \wedge time = j)$$

Here **EF** means that the statement will be true for some possible outcomes. We define this query to be $R_{EF}(i, BS, j)$.

The query to verify reachability between a sensor node i and BS at all time slots can be:

$$\forall j \ R_{AF}(i, BS, j)$$

For the constraint based reachability verification between a sensor node and the BS, we can generate the BDD using the algorithms described in Section 4, and reachability with cost constraint can be verified by the satisfiability of the BDD.

6. EVALUATION

To evaluate the performance and scalability of SensorChecker, we measured the time and space to build the transition BDD for the time slot based transition model. Note that the BDDs are reusable for multiple queries, and they can be built in an incremental way if there are some minor changes in the topology. If there are significant topology changes, then the BDDs need to be completely rebuilt. The data used to evaluate the SensorChecker is based on simulation, there is no need to use real data network because we need only forwarding and time schedule information to evaluate SensorChecker. Hence each real network can be used only once for evaluation purpose, it is hard to deploy different large real WSNs for evaluation and testing.

In our evaluation, all the evaluation examples were done on a machine with a 3G Intel Pentium IV CPU and 2G memory. For the time slot based transition model, we tested the BDD construction for our model in two types of topologies.

The first topology is a m by m mesh, where every square in the mesh contained a cluster of n nodes (Here m and n are adjustable parameters in the simulation). Every node was connected to any node that is in its neighbor square. The BS is located at a corner of the mesh. We tested the building overhead (time and space) of the transition BDD of the time slot based model for different square lengths (m), cluster sizes (n), number of awake nodes in a cluster, and ID encoding scheme. The second topology is a random network, where every node of the network is randomly deployed at a 1000 meters by 1000 meters square region and the radio range of every node is 50 meters. For this topology, we tested the building overhead (time and space) of the transition BDD for different number of nodes. Also, in all of our testing cases, the overhead to build reachability query and execute the query is negligible (less than 0.1 seconds) compared with the overhead to build the transition BDD, so we only consider the overhead (time and space) to build the transition BDD, which can be reused for multiple queries.

Impact of number of awake nodes on building transition BDD: Fig 2 and Fig 3 show the time and space needed to build the transition BDD for different m,n and different number of awake nodes in a cluster in a time slot. For the case of one awake node in a time slot, at any time slot i, if the intra-cluster ID of the node matches the value of (i *modula* cluster size), then the node is awake. For the case that half of the nodes in a cluster is awake in a time slot, at time slot i, if the intra-cluster ID of the node satisfies (i *modula* 2 = intra-cluster ID *modula* 2), then the node is awake.

We can see from Fig. 2 and Fig. 3 that the overhead increases with the increase of the cluster size and number of clusters. These results show that for regular topology, deterministic wake/dormant scheme, and deterministic ID encoding, our transition building algorithm is efficient. In the case of one awake node in a slot, when the cluster size increases from 24 to 32, the overhead does not increase very much or even decrease slightly. This is because 24 is not a power of 2 while 16 and 32 are, and BDD is good at manipulating quantities that are powers of 2. We can see that

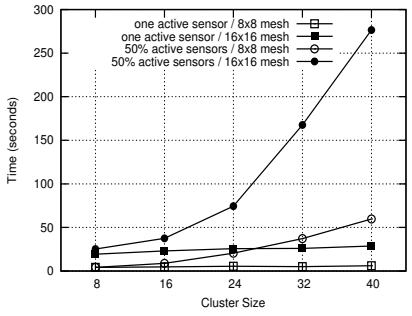

Figure 2: Time to build transition BDD with different number of awake nodes per slot

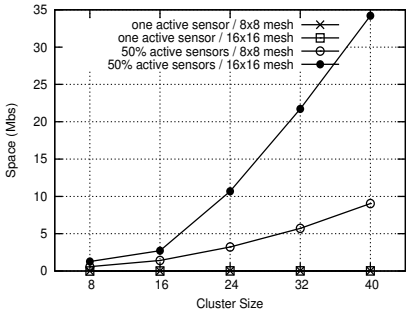

Figure 3: Space to build transition BDD with different number of awake nodes per slot

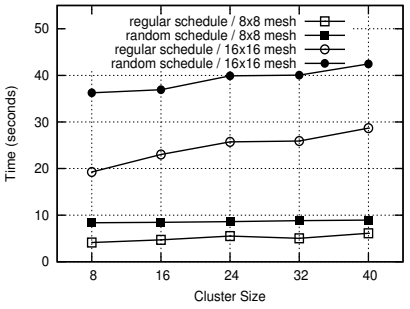

Figure 4: Time to build transition BDD with deterministic/random awake nodes

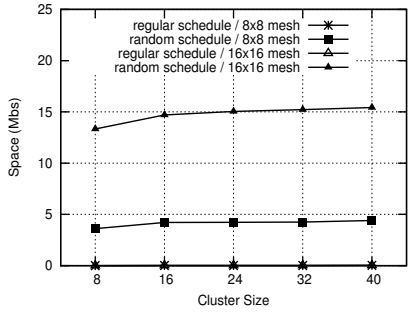

Figure 5: Space to build transition BDD with deterministic/random awake nodes

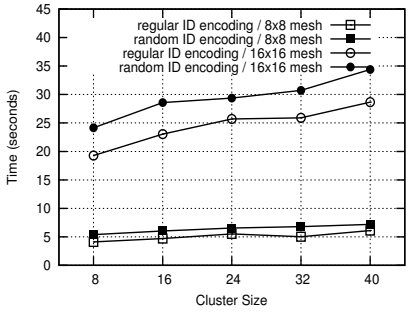

Figure 6: Time to build transition BDD with deterministic/random ID encoding

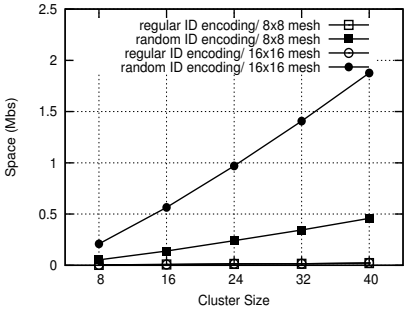

Figure 7: Space to build transition BDD with deterministic/random ID encoding

the overhead to build the transition is reasonable even for a sensor network with thousands of nodes. Also note that the BDD building is done at the base station, which is much more computationally powerful than the sensor nodes.

Impact of choices of awake nodes on building transition BDD : Fig. 4 and Fig. 5 show the impact of choices of awake nodes in the transition BDD building. In the simulation, there is the one awake node in every slot and the awake/dormant scheme is that a random node (uniformly distributed) in every cluster is awake during any time slot, compared with the case that a deterministic node is awake at every slot (that is, at any time slot i, if the intra-cluster ID of the node matches the value of i *modula* cluster size , then the node is awake). We can see that the time and space to build the transition in the random case is much higher than the deterministic case. This is because the transition BDD is essentially the compression of all information related to the topology, forwarding and awake/dormant schedule of the network, and the BDD can exploit the regularity in the deterministic case.

Impact of ID encoding scheme on building transition BDD: Fig. 6 and Fig. 7 show the time and space to build transition BDD for the deterministic ID encoding scheme (node ID is the concatenation of cluster ID and intra-cluster ID) versus the random ID encoding scheme (the ID of every node is a random 16-bit string). We can see that the overhead of building the transition with the random ID encoding scheme is higher than the case when deterministic ID encoding scheme is used.

Time overhead to build transition BDD in a random network: Fig. 8 and Fig. 9 show the time and space to build the transition BDD for a sensor network such that n nodes are randomly distributed over a 1000 by 1000 square, and the radio range of every sensor is 50. Every node is connected to any node that is in its radio range. We can see that the overhead increases about quadratically when the number of nodes increases. This is because that the number of edges increases quadratically when the number of nodes increases.

Time overhead to build BDD with cost constraints: For Algorithm 2, we tested the overhead to generate the BDD for the a 10 by 10 mesh, where every point in the mesh contains one node, and every node in the mesh is connected to all its neighboring nodes. Fig. 10 shows the time to build the BDD such that every node has a random cost between 0 and 20. The BDD represents the paths from the left bottom corner node to the right up corner node. We can see that the overhead decreases when the required maximum cost increases. This is because when the required maximum cost increases, the number of satisfiable paths decreases. Theoretically, Algorithm 2 is an exponential time algorithm. However, the algorithm is still practical if the number of satisfiable paths is tractable.

7. RELATED WORKS

Application of formal methods in sensor networks has not received much attention in the research community. A pro-

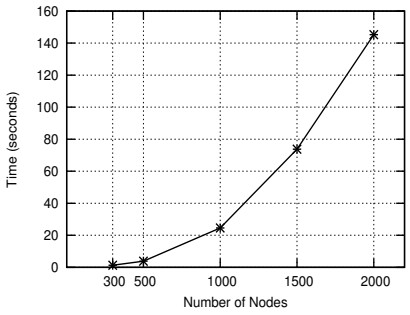

Figure 8: Time to build transition BDD for a random topology

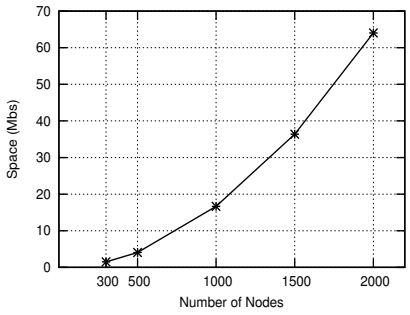

Figure 9: Space to build transition BDD for a random topology

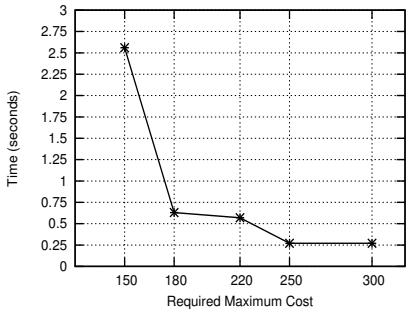

Figure 10: Time to build BDD with cost requirement for a mesh topology

cess algebra based formal language called PAWSN and an umbrella tool environment TEPAWSN that combines different formal techniques for modeling, analysis and development of power aware wireless sensor networks was presented in [7]. Modeling and verification of the LMAC protocols was discussed in [5]. All previous works address either specific protocols or some specific properties in the network. Therefore, we believe that SensorChecker is the first approach to apply formal methods in Mission-Oriented WSNs to provide reachability verification for general protocols in a rigorous and scalable way.

8. CONCLUSION

In this paper we propose formal models for automatic verification of reachability of Mission-Oriented WSNs. We model the end-to-end behavior of WSNs incorporating location, forwarding, awake/dormant schedule information using Binary Decision Diagrams (BBDs) and Symbolic Model Checking in order to verify the soundness and completeness of reachability properties. We then propose a technique to encode the topological path information using BDDs and verify constraint-based properties such as minimum cost reachability. The presented system is implemented in a tool called *SensorChecker* that is rigorously evaluated with various network sizes, topological and scheduling configuration. Our evaluation shows that our tool is scalable to thousands of sensors ($10K$ sensors for reachability) under different topological, and wake/dormant configurations.

9. REFERENCES

[1] Ehab Al-Shaer, Will Marrero, and Adle El-Atawy. Network configuration in a box: Towards end-to-end verification of network reachability and security. In *IEEE International Conference of Network Protocols (ICNP'2009)*, Oct. 2009.

[2] S. Chakravarty. A characterization of binary decision diagrams. *IEEE Trans. Comput.*, 42(2):129–137, 1993.

[3] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[4] E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 169–180, New York, NY, USA, 1982. ACM.

[5] A. Fehnker, L. Van Hoesel, and A. Mader. Modelling and verification of the lmac protocol for wireless sensor networks. In *Integrated Formal Methods*, pages 253–272. Springer, 2007.

[6] Taehyun Kim, Jaeho Kim, Sangshin Lee, Ilyeup Ahn, Minan Song, and Kwangho Won. An automatic protocol verification framework for the development of wireless sensor networks. In *TridentCom '08: Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, pages 1–5, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[7] K.L. Man, T. Krilavičius, Th. Vallee, and H.L. Leung. TEPAWSN: A formal analysis tool for wireless sensor networks. *International Journal of Research and Reviews in Computer Science*, 1(1), 2010.

[8] A. Rauzy. New algorithms for fault trees analysis. *Reliability Engineering and System Safety*, 40:203–211, 1993.

[9] P. Völgyesi, M. Maróti, S. Dóra, E. Osses, and Á. Lédeczi. Software composition and verification for sensor networks. *Sci. Comput. Program.*, 56(1-2):191–210, 2005.