SensorChecker: Reachability and Coverage Verification of Sensor Networks

***Abstract*—In recent years, Wireless Sensor Networks (WSNs) have gained considerable attention from researchers that made them very useful in various application domains. The wide deployment of WSNs in military and commercial applications has resulted in developing different protocols to support different WSN tasks. Sensors 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 mission. However, misconfigurations due to incorrect protocol specification or implementation might cause global malfunctioning in the reachability and coverage functions of WSNs.**

This paper presents novel techniques to statically verify the global reachability and coverage behavior of WSNs considering sensor configurations such as forwarding information and awake/dormant sensor 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 and the network coverage behavior of WSN based on sensor configuration using Binary Decision Diagrams (BDDs) and Symbolic Model Checking, and then define generic reachability and coverage properties using Computational Tree Logic (CTL). Second, we encode the WSN topological information using Boolean functions to verify constraint-based reachability and coverage properties for mission critical WSN, and show soundness and completeness.

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

I. Introduction

Wireless sensor networks (WSNs) have been increasingly used for large-scale data collection and event detection of critical applications such as hostile terrain and emergency responding where rapid deployment of network infrastructure is required [1], [2]. It is challenging to meet the reachability and coverage requirement necessary for achieving the mission of WSNs. First, most 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. 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 and coverage in WSNs, such as cost constraints in reachability, makes the verification more intractable. These characteristics of WSN 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.

Many protocols and algorithms have been proposed to configure sensor behavior in order to optimize routing and activity scheduling [3]–[6]. A realization of sensor configurations includes forwarding tables, actions (e.g., forward, encrypt, broadcast), and time slot activities (e.g., awake, dormant, etc). However, misconfigurations due to errors or bugs in the protocol specification or implementation might cause global malfunctioning that jeopardize reachability or coverage for the network missions. Examples of such problems include sensors unable to deliver data to the base station due to incomplete forward information, inconsistent time scheduling, or lack of coverage. Another example is that sensors can only deliver the information to the base station with high delay bound that might violate the information freshness in the application requirements.

In this paper, we present a novel approach to model and statically verify the global reachability and coverage behavior of a Wireless Sensor Network based on topology and configuration information used by WSNs. 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) [7] 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 and coverage properties of any WSN 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 [8] and richness of temporal logic such as Computational Tree Logic (CTL) [9] can facilitate reachability and coverage verification. Our system is implemented in a tool called SensorChecker which is evaluated rigorously with different WSN sizes and topologies.

As shown in Figure 1, SensorChecker is an off-line tool and it does not interfere with the operations of deployed WSNs. It basically reads the topological and configuration information of a WSN as an input and allows users to generate any CTL-based query to verify the future reachability and coverage behavior of the system. If the query is unsatisfied, the

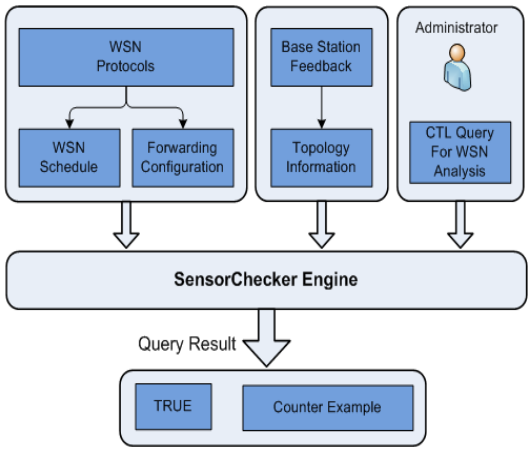

Fig. 1. SensorChecker System Architecture

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 has been proposed to verify sensor network protocol specifications [10]–[14]. However, unlike SensorChecker, these techniques are limited to specific protocols and properties. Other approaches were proposed to verify network security configuration such as ConfigChecker [15]. 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. Also, ConfigChecker does not support constraint-based reachability queries. To the best of our knowledge, we are unaware of any other work that addresses global reachability and coverage verification of WSN configuration considering these challenges.

The structure of SensorChecker is described in Fig 1. The input of the engine includes three parts: the topology information, which can be obtained from Base Station, the WSN schedule and packet forwarding configuration as determined by WSN protocols and algorithms, and finally the CTL-based user Queries.

The remaining sections of the paper are organized as follows: Section II presents our system assumptions; Section III discusses the state machine based model for wireless sensor networks; Section IV discusses the constraint-based property verification for reachability and reliability; Section V discusses CTL based reachability and coverage verification in sensor networks; Section VII presents the main implementation technique of SensorChecker; Section VIII is the evaluation results; Section IX is the related work; and Section X presents the conclusion and future work.

II. System Assumptions for Sensor Networks

We assume a typical sensor network where sensors 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. This can be achieved by any location determination system for sensor networks [16].

To avoid collisions and to save energy, a sensor node may be on sleep mode for some time slots. Thus sensor node has two possible states: awake or dormant. An awake node is able to sense an event, transmit a packet or receive a packet. A dormant node turns all its function modules off except a timer to wake itself up. All nodes have their own awake/dormant schedules. These schedules are shared with neighboring nodes. Therefore, nodes are locally synchronized. For sensing purposes, the working schedules of sensor nodes are normally periodic [17], [18]. However, our verification model can accommodate arbitrary awake/dormant and sensing schedules, which are abstracted as information in the node's configuration table.

We also assume that every sensor node can act as a router, and an arbitrary multi-hop routing protocol for sensor networks has been running in the network so that each node knows how to forward the packets it receives. Each routing protocol configuration is abstracted in our model as information stored in the configuration table of each sensor node. In the routing table of a sensor node, there may exist multiple routing entries for a given destination. This is necessary for any awake/dormant scheme, such that a node should choose the appropriate next hop based on the awake/dormant status of the neighbors.

In our model, we assumed that a sensor 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 it and one of its next hops are awake. Our model is general enough to include different classes of protocols including traditional topology based routing protocols [19], [20], data-centric protocols [21]–[23], hierarchical protocols [24], location-based routing protocols [25], [26], flooding/ broadcast [27] or multicast-based protocols, among others. The model can also capture extensions for reliability support, for example multipath routing as we show in the rest of the paper.

Before the reachability and coverage 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 sensor nodes from sensors 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 sensors to the Base Station. For the packet forwarding scheduling information, depending on the routing protocol this can be achieved by different ways. 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.

In the presented model, we ignored the cases of retransmissions. We rely on the redundancy of sensing data by other sensors as an event may be reported my multiple sensors.

The reachability and coverage 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 sensor to check if a specific reachability or coverage requirement is enforced or not, based on the results, protocol bugs or sensors misconfiguration will be reported.

III. Basic Model of WSN Configuration Transitions

A. Time Slot Based State Representation

We can model the sensor network as a finite state machine such that the state transitions represent information flow from one node to another node. After receiving a packet, a sensor 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:

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

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 by evaluating to true whenever the parameters used as input to the function correspond to a packet that is in the network and false otherwise. 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 sensing, receiving and forwarding. We have encoded the dormant status as 0 and awake status as 1.

B. Building State Transitions

State transitions in the model correspond to packets moving between sensors in the network. Our goal here is to build a single BDD for the whole network. 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.

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 $j \geq i$;
        $T = T \vee ((\textbf{Src} = s \wedge \textbf{Dest} = d \wedge \textbf{Loc} = x \wedge \textbf{Time} = i) \wedge (\textbf{Src}' = s \wedge \textbf{Dest}' = d \wedge \textbf{Loc}' = k \wedge \textbf{Time}' = j + 1))$;
      end
    end
  end
  $i = i + 1$;
end

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. **Dest**$'$, **Loc**$'$, **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 of the next hop that leads to the destinations 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 variables are used:

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.

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:

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

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.

IV. 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 2 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.

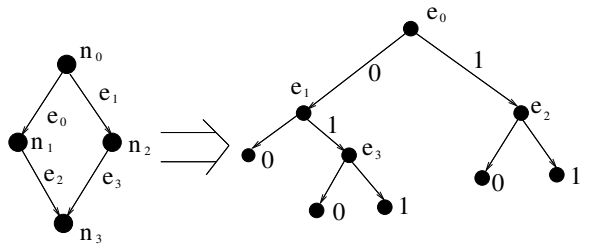

Fig. 2. BDD representation of paths in a graph

The algorithm to generate the BDD is shown in Algorithm 2, which is an extension of the fault tree analysis algorithms in [28].

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. The general problem of finding paths satisfying multiple requirements is NP-hard [29]. To reduce search space, Algorithm 2 applies branch pruning during the search process. As long as the number of satisfiable paths is tractable, the algorithm is practical. One limitation of the algorithm is that a single BDD can only represent paths between a specified source and destination pair. However, in sensor networks, the most common traffic is between Base Station and sensors, so the algorithm can be applied to check constraint based reachability between the Base Station and any sensor in consideration.

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;
```

V. WSN Configuration Verification Using Symbolic Model Checking

To facilitate configuration verification in Wireless Sensor Networks, we use Computational Tree Logic (CTL) [9] 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 realized. The following categories of CTL operators are used:

- Boolean Operators
- CTL operators, which includes **EX**, **EF**,**AX**, **AF**, **AU**,**EU**, **AG**,**EG**
- Some extended CTL operators, for example, bounded or reverse time operators

In this paper we focus on queries related to reachability and coverage.

A. Basic Reachability Verification

In this subsection we describe how to construct a query BDD related to reachability. First we define soundness and completeness of reachability in sensor networks.

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 $R_1(i, BS, j)$.

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 $R_2(i, BS, j)$.

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

$$\forall j \ R_1(i, BS, j)$$

Now we define soundness of reachability.

Definition 1. A WSN configuration, $\mathcal{C}$, is *sound* if only those authorized sensors can reach BS at some time slot j.

The query of soundness can be defined as:

$$R_2(i, BS, j) \rightarrow authorized(i)$$

This means that only authorized nodes can reach BS for some specified time slots. Authorized nodes are all legitimate and none suspicious nodes in the WSN.

Next we define completeness of reachability.

Definition 2. A WSN configuration, $\mathcal{C}$, is *complete* if all those authorized sensors can reach BS at some time slot j.

The query to verify completeness of reachability can be:

$$\forall i \ authorized(i) \rightarrow \exists j \ R_1(i, BS, j)$$

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

If we want some resilience of the path between source and destination, for example, we want the reachability still holds when a node or a group of nodes are removed or compromised, we can set status bit of a node to be "dormant" permanently.

To verify node i can reach BS if a specific node j fails:

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

Multi-path routing, where more than one non-intersecting routing path can be used between the source and destination, to enhance both the reliability and security of sensor networks. To guarantee that there exists at least two node-disjoint paths from node i to BS, we can use the following query.

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

That is, there is no bridge node between the source (node i) and destination (BS). This is equivalent to the existence of two node-disjoint paths between source and destination.

Proposition 1: The above query will verify if there are two node disjoint paths from the source to BS.

Proof: If there exists two node disjoint paths from a node i to BS, then it clear that to see the node can still reach the BS after removing any node other than i and BS. On the other hand, if node i can reach BS after removing any node j other than i and BS, then there is no bridge node between i and BS; which means there are at least two node disjoint paths from i and BS. ■

To verify that two paths exist between a node i and BS with certain constraints, we can build a BDD that represents the satisfying paths and check that after the removal of any node other than i and BS if the BDD is still satisfiable. The removal of any node in the BDD can be done by the BDD restriction operation [30].

To guarantee that there exist at least two braided (edge-disjoint) paths from node i to BS, we need to slightly modify the transition model. We just add a status bit for every edge in the transition, so we can remove any edge by setting the status bit of the edge to be 0.

The query looks like:

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

That is, the removal of any edge will not affect the reachability; this is equivalent to the existence of two edge-disjoint paths between node i and BS.

VI. Coverage Verification

A. Basic Coverage Verification

In many WSNs, it is required that the whole region in consideration to be covered by sensors. Before the discussion of applying formal methods in coverage verification, we need to define coverage formally. Suppose the region to be monitored

by the WSN is R, which contains non-overlapping subregions $R_1, R_2, \ldots, R_n$. That is,

$$R = \bigcup_{1 \le i \le n} R_i$$

Suppose that there are m sensors in the WSN and every sensor covers a certain set of subregions. We define basic coverage as follows:

At any time slot, the set of all wake sensors can collectively cover the entire area R.

Formally, the above statement is equivalent to the following first order logic statement:

$$\forall i, u \, \exists v \; covers(S_v, R_i) \wedge awake(S_v, t_u)$$

Here $covers(S_v, R_i)$ means that sensor S_v covers subregion R_i, $awake(S_v, t_u)$ means sensor S_v is awake at time slot t_u.

Suppose we are considering a total of 2^l slots, then we can represent the 2^l slots in l bits $t_0, t_1, \ldots, t_{l-1}$, where t_0 is the least significant bit. For example, if $l = 2$ and a sensor is awake at time slots 0 and 2, then the awake slots of this sensor can be represented as $T = (\overline{t_0} \wedge \overline{t_1}) \vee (\overline{t_0} \wedge t_1)$, which can be simplified as $T = \overline{t_0}$.

For every subregion R_i, its coverage can be represented as:

$$C_i = \bigvee_{v \in \{v | S_v \text{ covers } R_i\}} T_v$$

Here T_v is the awake time slots representation of sensor S_v.

For the whole area R, the coverage can be represented as:

$$C = \bigwedge_{1 \le i \le n} C_i$$

Note that C_i, C_i, and C are all BDDs of variables $t_0, t_1, \ldots, t_{l-1}$. To verify the coverage of a WSN, we need to check the validity of C. We have the following Proposition.

Proposition 2: The validity of C is equivalent to the coverage of the WSN.

Proof:

If C is valid, then all C_i are valid, which means all R_i is covered at all time slots.

On the other hand, if C is not valid, then at least one of the C_i for some i is not valid, which in turn means that there exist some assignments of $t_0, t_1, \ldots, t_{l-1}$ such that C_i is not true. This is equivalent to that there exist at least one time slot at which the subregion R_i is not covered, which in turn means that R is not covered.

Now we can see that the validity of C is equivalent to the coverage of the WSN.

■

To verify the coverage at a certain time period between T_1 and T_2, the following query can be used:

$$\forall (time\ intervals\ between\ T_1\ and\ T_2) \forall (subregion\ R_i) \exists (awake\ node\ S_v)(\ cover(S_v, R_i))$$

This query can be checked by the validity of C when $t_0, t_1, \ldots, t_{l-1}$ are assigned values between T_1 and T_2. This can be stated formally as:

$$\bigwedge_{i \in [T_1, T_2]} C|_{t_0, t_1, \ldots, t_{l-1} \text{ is the binary representation of } i}$$

In some applications, we require that the monitored regions need to covered by at least k sensors at all times. For example, one needs to have the region to be covered by at least 3 sensors at all times if triangulation method is used to determine the exact location of the monitored event. Also, k-coverage can improve the reliability of the WSN.

For every subregion R_i, its k-coverage can be represented as:

$$K_i = \bigwedge_{1 \le u \le 2^l} \bigvee_{n_1, \ldots, n_k \in \{ \text{ sensors cover } R_i\}} \left(\bigwedge_{1 \le w \le k} awake(n_w, u) \right)$$

Here $awake(n_w, u)$ is the awake/dormant status of sensor S_{n_w} at time slot u. If S_{n_w} is awake at slot u, then $awake(n_w, u)$ is 1, otherwise it is 0.

For the whole area R, the k-coverage (denoted as K) can be represented as:

$$K = \bigwedge_{1 \le i \le n} K_i$$

If we need to verify that every subregion R_i should be covered once for any consecutive $d + 1$ time slots, we can represent the coverage as:

$$W_i = \bigwedge_{1 \le u \le 2^l} \bigvee_{v \in \{v | S_v \text{ covers } R_i\}} \bigvee_{0 \le j \le d} awake(S_v, u + j)$$

For the whole area R, this coverage (denoted as W) can be represented as:

$$W = \bigwedge_{1 \le i \le n} W_i$$

VII. Implementation

To implement Algorithm 1, we use 16 bits to encode the ID of the nodes, 10 bits to encode the index of time slots. The ID of any node is the concatenation of the cluster ID (10 bits) and an intra-cluster ID (6 bits). So in total we need 52 bits to encode a state; which includes the source/destination ID and the index of starting time slot and ending time slots. By keeping track of the states and the transition relation implicitly via formulas represented using BDDs, it is possible to perform analysis on our model. A BDD represents a decision graph where the nodes in the graph are vertices and the edges coming out of a vertex represent the two possible Boolean assignments to that variable. Thus, a complete assignment to all variables corresponds to a path in the graph which ends in a value of true or false, which is the value of the formula when given that assignment. The size of the BDD is sensitive to the order in which the variables appear in the decision graph. In our implementation, we use the common interleaving heuristic which

places the next state copy of a variable right next to the current state copy of the variable in the ordering. For example, suppose the variables used for ID are $x_0, x_1, \ldots, x_{15}$, the variables used for time slot index are $y_0, y_1, \ldots, y_9$, and the variables used for the next state ID are $x'_0, x'_1, \ldots, x'_{15}$, the variables used for next time slot index are $y'_0, y'_1, \ldots, y'_9$, we use the ordering $x_0, x'_0, x_1, x'_1, \ldots, x_{15}, x'_{15}, y_0, y'_0, y_1, y'_1, \ldots, y_9, y'_9$. For our implementation, we are using BuDDy2.2 package [30] which is a well known package that implements BDD operations and memory management. The package is implemented in C, with an added encapsulation for C++.

VIII. 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, and the coverage BDD for coverage verification. 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 sensors (Here m and n are adjustable parameters in the simulation). Every sensor was connected to any sensor 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 in transition BDD building: Fig 3 and Fig 4 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 3 and Fig 4 that the overhead increases with the increase of the cluster size and number of clusters. This 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 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 in transition BDD building: Fig. 5 and Fig. 6 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 in transition BDD building: Fig. 7 and Fig. 8 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.

Transition BDD building in a random network: Fig. 9 and Fig. 10 show the time 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 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

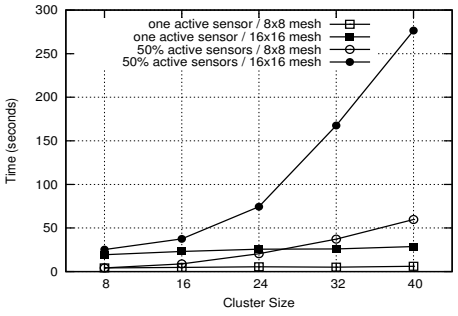

Fig. 3. Time to build transition BDD with different number of awake nodes per slot

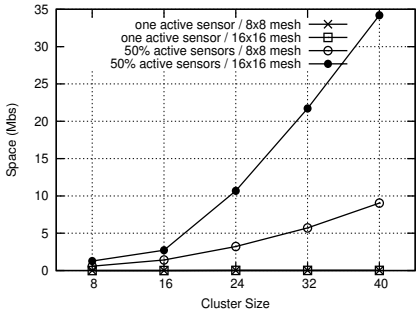

Fig. 4. Space to build transition BDD with different number of awake nodes per slot

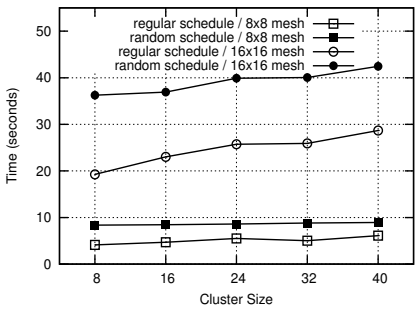

Fig. 5. Time to build transition BDD with deterministic/random awake nodes

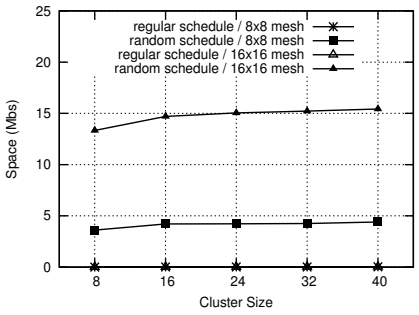

Fig. 6. Space to build transition BDD with deterministic/random awake nodes

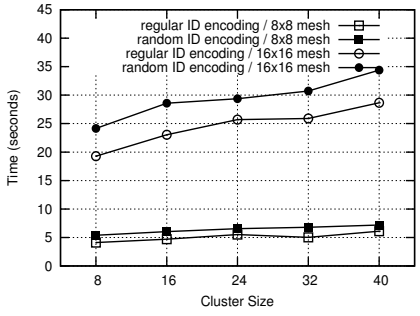

Fig. 7. Time to build transition BDD with deterministic/random ID encoding

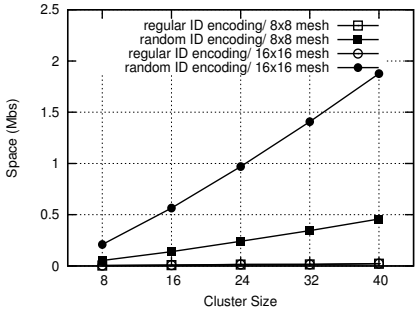

Fig. 8. Space to build transition BDD with deterministic/random ID encoding

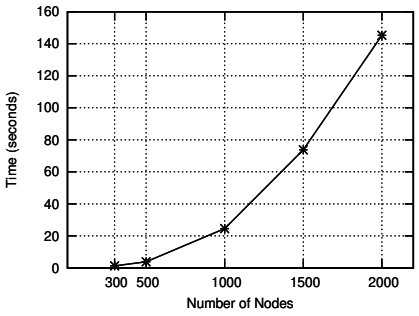

Fig. 9. Time to build transition BDD for a random topology

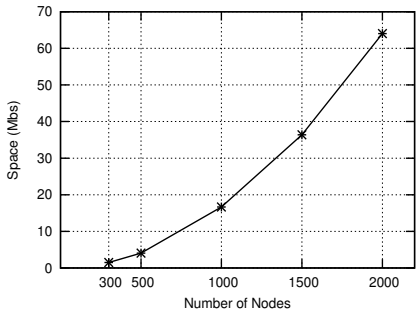

Fig. 10. Space to build transition BDD for a random topology

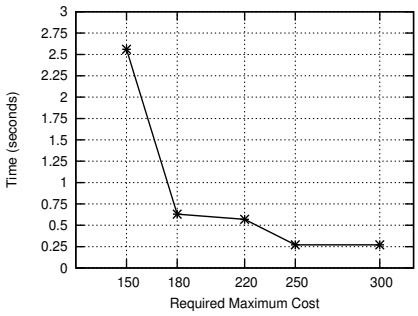

Fig. 11. Time to build BDD with cost requirement for a mesh topology

one node, and every node in the mesh is connected to all its neighboring nodes. Fig. 11 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.

Coverage Verification: We also tested coverage verification. The main overhead in coverage verification is to build the coverage BDD C. We generated a m by m mesh topology, where every square is a subregion and contains a sensor, and every sensor covers the square that it occupies and the four neighboring squares. There are l time slots, and every sensor will be awake with probability 0.5 in any time slot. The following table shows the time needed for coverage verification.

m	No. of sensors	l	time(seconds)
50	2500	1024	0.09
100	10000	1024	0.35
200	40000	1024	1.41
50	2500	4096	0.34
100	10000	4096	1.44
200	40000	4096	5.73

We also tested 3-coverage verification, the following table shows the verification time.

m	No. of sensors	l	time(seconds)
50	2500	1024	0.62
100	10000	1024	2.55
200	40000	1024	11.28
50	2500	4096	2.83
100	10000	4096	12.07
200	40000	4096	51.26

We can see that the verification time is almost linear in the number of nodes and number of time slots. This shows that the coverage verification method is scalable.

IX. Related Works

Many of the current literature has addressed network configuration as a challenging problem. In [31], the authors identify all anomalies that could exist in a single- or multi-firewall environment, and present a set of techniques and algorithms distributed legacy firewalls. These techniques are implemented in a software tool called the "Firewall Policy Advisor" that simplifies to automatically discover policy anomalies in centralized and the management of filtering rules and maintains the security of firewalls. In [32], a generic model that captures various filtering policy semantics using Boolean expressions is presented.

The model can be used to derive a canonical representation for IPSec policies using Ordered Binary Decision Diagrams (OBDD). Based on this representation, a comprehensive framework is developed to classify and identify conflicts that could exist in a single IPSec device (intra-policy conflicts) or between different IPSec devices. In [33], a static analysis toolkit called FIREMAN, is proposed for firewall modeling and analysis. This toolkit treats firewall configurations as specialized programs and applies static analysis techniques to check misconfigurations, such as policy violations, inconsistencies, and inefficiencies, in individual firewalls as well as among distributed firewalls. All these approaches address the problem of configuration conflicts in Firewall and IPSec devices.

Other studies targeted analyzing routing configuration. The first quantitative study of BGP misconfiguration was done in [34]. In [35], a router configuration tool called *rcc* was proposed. Tool *rcc* can find faults in BGP configuration using static analysis. In [36], the authors showed that determination of IBGP configuration correctness is NP-hard, but some simple sufficient conditions on network configurations can guarantee correctness. In [37], the authors presented and evaluate the idea of shadow configurations, which allow configuration evaluation before the deployment and thus can reduce potential network disruptions.

Anquiro [38] is a domain specific model checker for static verification. It is used to check WSN software written in Contiki C. In Anquiro, the properties are specified using Linear Temporal Logic (LTL) leveraging Bogor model checker. Anquiro provides different abstraction levels, the system-wide communication abstraction implements the effect of multi-hop communication. However, this abstraction can check reachability between base station and sensors but can not verify the coverage.

A novel end-to-end network configuration analysis system for IP based networks was proposed in [15] and implemented in ConfigChecker. However ConfigChecker cannot be applied to WSNs, while SensorChecker establishes new models for time scheduling and packet forwarding in sensor networks, and can verify constraint based reachability and coverage in WSNs.

Application of formal methods in sensor networks has not received much attention in the research community. A process 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 [12]. Software verification for sensor networks was discussed in [11]. A Framework consists of a test procedure description language written in XML was proposed in [10]. Formal modeling and analysis of wireless sensor network algorithms in Real-Time Maude was discussed in [13]. Modeling and verification of the LMAC protocols was discussed in [14]. All previous works address either specific protocols or some specific properties in WSNs. Therefore, we believe that SensorChecker is the first approach to apply formal methods in WSNs to provide reachability and coverage verification for general protocols in a rigorous and scalable way.

X. Conclusion and Future Work

The deployment success of any WSN is dependent on the ability of its configuration to satisfy the reachability and coverage. In this paper we propose formal models for automatic verification of reachability and coverage of WSNs. As usually the case in such formal approaches, the key challenge is to create an accurate and scalable abstraction model for the system. This is particularly important for WSN as it usually exhibits a large number of sensors with arbitrary location and scheduling configurations. 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 and coverage 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 coverage is verified to guarantee that a designated area can be completely covered by at least k sensors during T time periods.

Previous works in formal network configuration verification is either limited on specific WSN protocols or not applicable on WSN properties such as schedule-constrained coverage and 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 and $40K$ sensor for coverage) under different topological, and wake/dormant configurations. In the future, we plan to extend the models

to handle safety and security properties. In addition, we will also investigate configuration planning problem ($i.e.$, how to generate valid configuration from given requirements).

REFERENCES

[1] K. Lorincz, D. Malan, T. R. F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, and M. Welsh, "Sensor networks for emergency response: Challenges and opportunities," *IEEE Pervasive Computing, Special Issue on Pervasive Computing for First Response*, Oct-Dec 2004.

[2] D. Malan, T. Fulford-jones, M. Welsh, and S. Moulton, "Codeblue: An ad hoc sensor network infrastructure for emergency medical care," in *In International Workshop on Wearable and Implantable Body Sensor Networks*, 2004.

[3] C. P. Singh, O. P. Vyas, and M. K. Tiwari, "An overview of routing protocols of sensor networks," *Computational Intelligence for Modelling, Control and Automation, International Conference on*, vol. 0, pp. 873–878, 2008.

[4] T. Yardibi and E. Karasan, "A distributed activity scheduling algorithm for wireless sensor networks with partial coverage," *Wirel. Netw.*, vol. 16, no. 1, pp. 213–225, 2010.

[5] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325–349, 2005.

[6] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, p. 4, IEEE Press, 2005.

[7] S. Chakravarty, "A characterization of binary decision diagrams," *IEEE Trans. Comput.*, vol. 42, no. 2, pp. 129–137, 1993.

[8] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model checking*. Cambridge, MA, USA: MIT Press, 1999.

[9] E. A. Emerson and J. 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*, (New York, NY, USA), pp. 169–180, ACM, 1982.

[10] T. Kim, J. Kim, S. Lee, I. Ahn, M. Song, and K. 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*, (ICST, Brussels, Belgium, Belgium), pp. 1–5, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[11] P. Völgyesi, M. Maróti, S. Dóra, E. Osses, and A. Lédeczi, "Software composition and verification for sensor networks," *Sci. Comput. Program.*, vol. 56, no. 1-2, pp. 191–210, 2005.

[12] K. Man, T. Krilavičius, T. Vallee, and H. Leung, "TEPAWSN: A formal analysis tool for wireless sensor networks," *International Journal of Research and Reviews in Computer Science*, vol. 1, no. 1, 2010.

[13] P. Olveczky and S. Thorvaldsen, "Formal modeling and analysis of wireless sensor network algorithms in Real-Time Maude," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, p. 8, 2006.

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

[15] E. Al-Shaer, W. Marrero, and A. 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.

[16] A. Youssef and M. Youssef, "A taxonomy of localization schemes for wireless sensor networks," in *The International Conference on Wireless Networks*, 2007.

[17] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 214–226, ACM, 2004.

[18] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, *et al.*, "A macroscope in the redwoods," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, p. 63, ACM, 2005.

[19] M. Younis, M. Youssef, and K. Arisha, "Energy-aware routing in cluster-based sensor networks," in *10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings*, pp. 129–136, 2002.

[20] J. Chang and L. Tassiulas, "Maximum lifetime routing in wireless sensor networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 12, no. 4, pp. 609–619, 2004.

[21] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pp. 174–185, ACM, 1999.

[22] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 56–67, ACM, 2000.

[23] D. Braginsky and D. Estrin, "Rumor routing algorthim for sensor networks," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, p. 31, ACM, 2002.

[24] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 2000*, p. 10, 2000.

[25] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, p. 84, ACM, 2001.

[26] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," *UCLA Computer Science Department Technical Report, UCLA-CSD TR-01-0023*, 2001.

[27] S. Hedetniemi, S. Hedetniemi, and A. Liestman, "A survey of gossiping and broadcasting in communication networks," *Networks*, vol. 18, no. 4, pp. 319–349, 1988.

[28] A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering and System Safety*, vol. 40, pp. 203–211, 1993.

[29] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[30] J. Lind-Nielsen, "The BuDDy OBDD package." http://www.bdd-portal.org/buddy.html.

[31] E. S. Al-shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *In IEEE INFOCOM '04*, pp. 2605–2616, 2004.

[32] H. Hamed, E. Al-Shaer, and W. Marrero, "Modeling and verification of ipsec and vpn security policies," in *ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols*, pp. 259–278, 2005.

[33] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, and P. Mohapatra, "FIREMAN: A toolkit for firewall modeling and analysis," in *IEEE Symposium on Security and Privacy (SSP'06)*, May 2006.

[34] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 3–16, ACM, 2002.

[35] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *NSDI*, 2005.

[36] T. G. Griffin and G. Wilfong, "On the correctness of IBGP configuration," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 17–29, ACM, 2002.

[37] R. Alimi, Y. Wang, and Y. R. Yang, "Shadow configuration as a network management primitive," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 111–122, 2008.

[38] L. Mottola, T. Voigt, F. Österlind, J. Eriksson, L. Baresi, and C. Ghezzi, "Anquiro : Enabling Efficient Static Verification of Sensor Network Software," in *1st International Workshop on Software Engineering for Sensor Networks (SESENA - colocated with 32nd ACM/IEEE International Conference on Software Engineering ICSE)*, 2010.