Guided Simulation of a Wireless Sensor Network

Petr Sobeslavsky*
Supervisor: Laurent Mounier*
*Verimag, 2 avenue de Vignate, F-38610 Gières, France
Email: petr@sobeslavsky.net

***Abstract*—Guided simulation is a technique that allows a developer to guide a simulation of a complex system towards a certain goal. In this paper we explore the possibility of using guided simulation as a tool for development of new algorithms in wireless sensor networks. We propose a new strategy for efficient exploration of the state space and show that the strategy is feasible.**

***Index Terms*—Guided simulation, model checking, simulation, wireless sensor networks**

I. Introduction

After years of research, wireless sensor networks (WSNs) matured to a state where they are ready for practical applications. However, some issues still remain unresolved. One of them is securing such networks. New solutions addressing the problem are currently under development.

An important characteristic of a wireless sensor network is the limited computational power of the nodes and limited capacity of their batteries. A good solution ensures the required level of security and uses as little resources as possible. So one of the challenges in designing such a solution is finding a good tradeoff between security and performance.

When looking for the tradeoff, it might be useful for a designer to be able to find the least efficient execution sequences easily and use the knowledge to further improve their algorithm.

The objective of our work was the following: given a non-deterministic operational model of a network, find a way to find the execution sequence which is the least efficient with respect to a given metric.

We proceeded the following way: First, we reviewed the possible directions and summarized our findings in Section II of this paper. Then, we implemented a simulation system adjusted for our field of research (described in Section III) and a model of a wireless sensor network (Section IV). In Section V we present our experimental results. Section VI concludes the paper.

II. Related Work

There are two traditional ways of evaluating properties of a model - simulation and model checking. Recently, a new class of techniques, called guided simulation, has been proposed. This section reviews all the three approaches.

A. Simulation

When using simulation, the network nodes are implemented in some standard network simulation environment (e.g. NS-2 [1], OMNeT++ [2]) and a set of runs using different scenarios is performed. After running sufficient number of simulations the obtained values are evaluated and a conclusion stated.

The advantage is that in these tools it is possible to simulate the nodes with a high level of detail, including the MAC layer with packet collisions. It might also be possible to directly use source codes from an existing implementation, without needing to translate them into a simulation language.

The disadvantage is that the simulation results depend on the specific simulation run and on the specific choice of random numbers which determine the simulation. Results obtained after a number of simulation runs would approximate the average case scenario, but it would not be possible to use them to prove any statement about the worst-case scenario.

B. Model Checking

In order to use model checking, the network has to be described in a special language, which is then compiled into a finite state automaton. A model checking tool generates all possible execution sequences and chooses the ones with interesting properties as an output.

The advantage is that model checking gives a proof of the result. It might be used not only to find the average case, but also to find the provably worst case scenario.

The disadvantage is that the size of the state space increases exponentially and it is not feasible to simulate networks of size of more than several nodes, which is not sufficient for evaluation of complex security solutions.

C. Guided Simulation

In [4] a spectrum of techniques between simulation and formal verification has been proposed (Fig. 1). Right next to the simulation is the simulation extended with *coverage measures* which denote how many and what kinds of states have been explored. Further to the right is a *smart simulation*, which generates functional tests based on a coverage metric, driven by the desire to cover an abstract state machine or selected corner states. Next class of techniques is a *wide*

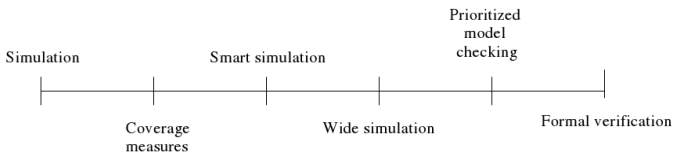

Fig. 1. A spectrum of verification techniques as defined in [4]

simulation which symbolically represents large sets of states as Binary Decision Diagrams (BDD) or logical expressions. The last class, just next to the formal verification, is *prioritized model checking*. The idea is to explore the state space in parts using heuristics.

An approach which would be on the right of the spectrum, called *guided search*, has been further explored in [11]. The goal was to optimize model checking for bug finding by using heuristics to search the part of the state space that is most likely to contain design flaws. Several heuristics were considered:

- *Target Enlargement* Error states are enlarged so they can be found with less searching.
- *Hamming Distance* A distance to a goal state is used as a search metric.
- *Tracks* Approximate preimages of states on a desired execution sequence are automatically defined and enlarged using the Target Enlargement technique.
- *Guidedposts* A designer is supposed to provide a series of conditions that he believes to be interesting or even required preconditions.

In [5] a combination of random and symbolic simulation is considered. The simulator converts input signals into symbolic formulas and stores them as Binary Decision Diagrams. A SAT solver is then used to search for a new coverage state.

A combination of heuristics with random simulation has been proposed in [10]. The Guido uses and abstraction engine to work only with an abstraction of the complex model. The abstract machine is then used to guide the simulation towards the goal. It is done using a heuristic function, which tries to minimize the distance to the goal.

After reviewing existing proposals, the authors in [8] argue that heuristics using greedy strategies tend to get stuck in local optima. Just expanding the search horizon adds too much search overhead for limited ability to escape dead-ends. They propose a guidance strategy, which remembers multiple states from which to continue search. It keeps a bucket of states for each onion ring distance from the goal. The exploration algorithm then tries to find a balance between greed and relaxation.

III. IfSim Simulation Architecture

In order to experiment with various simulation strategies and compare the results. Following the previous Verimag experience [7], we used the IF Toolset and developed a simulation library *IfSim*. It is a universal library and can be used to simulate any model described in the IF language.

A. If Toolset

The IF Toolset [3] is an environment for modelling and validation of heterogeneous real-time systems. It provides the IF language to describe the model and the IF engine to execute it.

1) IF Language: The IF language is a rich formalism for structured automata-based system representations. An IF specification consists of a set of processes, where process instances run in parallel and communicate either through shared variables or message passing.

A process is a time automaton extended with data. Each process instance owns a set of variables and a FIFO queue to store incoming messages. A process can move from one state to another by executing a transition. A transition can be triggered by a message reception, by a timer reaching a value or when a condition on variable values holds. Transition is a sequence of actions which are executed sequentially and comprise variable assignments, message sending, process creation etc.

The execution of a transition is instantaneous and time progresses only between the transitions. Time distances are measured by variables of type clock, which can be changed or reset in a transition.

2) IF Engine: The IF engine translates the IF code into a source code in C++ and compiles it to an executable file. The IF engine itself is also written in C++ and enables users to extend its functionality by linking their own libraries.

A user has also access to the low level functions of the engine and is able to implement new exploration algorithms.

Unfortunately, the current IF implementation does not release memory used by already explored states, so the number of states explored is limited by the capacity of the memory. There is no support for such an operation and due to the complexity of the internal data structures it was not possible to implement it properly. We partly solved the problem by introducing reference counting on the instances of the state class, but there are still some objects that stay unreleased and the number of states the engine can explore in one run is still limited.

B. Simulator Library

The execution of the model in the IF engine is driven by the *IfDriver* class. The *IfSim* simulator library defines its own structure of classes derived from the *IfDriver*. (Fig. 2)

There are two essential components:

- *Evaluator* For a given state, evaluates the value of one (*MetricEvaluator*) or several (*GuidedEvaluator*) metrics and stores them in the state. It also provides a function which decides whether a state is a goal state or not. A user is supposed to provide a class derived from the evaluator class which will implement the functions specific for his model.
- *Explorator* Explorator is the implementation of an exploration algorithm. Several explorators are provided in *IfSim* and described in the following subsection. The user can simply instantiate an explorator class and run the simulation.

C. Exploration algorithms

We implemented several exploration algorithms. All of the algorithms assume that for each state it is possible to assign a non-negative real value of a metric and that an assignment function is provided by the user. If the user provides values of

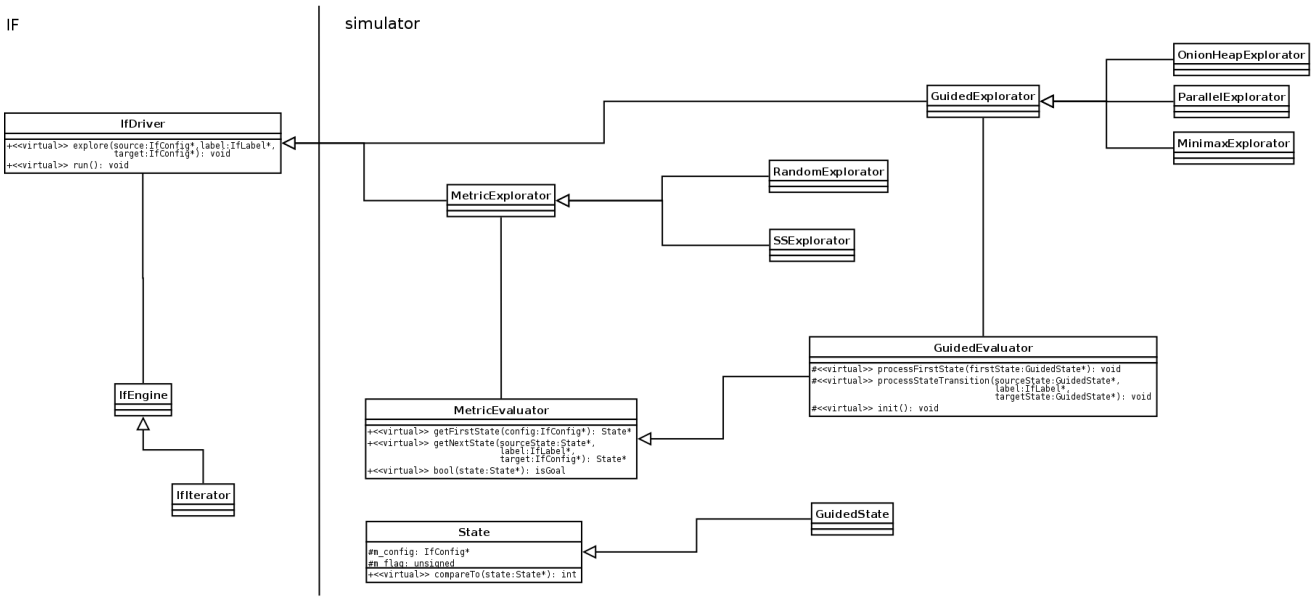

Fig. 2. IfSim class diagram

several weighted metrics for a state, their weighted harmonic mean is used.

The implemented algorithms are:

1) Random Exploration: In every state the next step is chosen randomly among all the subsequent states. This exploration corresponds to a standard simulation (with the limitation that the probability for all the states is equal).

2) Heap Exploration: Tries to find the shortest-sequence execution by keeping all the unexplored states in a priority heap ordered by the value of the metric. At each step, the state from the top of the heap is taken, all its subsequent states are evaluated and put on the heap. In order to limit the state explosion, the upper limit on the size of the heap can be set. When the heap gets too big, the states at the bottom of the heap are deleted.

3) Onion Heap Exploration: The idea of the algorithm comes from the onion rings [8]. The simulator creates a priority heap for each layer of states with the same distance from the source. Every step, one of the heaps is chosen, the state on the top is evaluated and all its subsequent states put on the heap in the next layer.

The heaps are ordered by the value of the metric. The heap selection starts with the heap which is the closest to the initial state. A coin is flipped. If the result is positive, the heap is used, otherwise the next heap is tried, etc. If an empty heap is selected, the next heap is used.

4) Parallel Exploration: The algorithm chooses a fixed number of execution sequences, which cover the whole spectrum of the metric value and runs them in parallel. At each step, all the consequent states of the current state are evaluated and ordered by the value of the metric. The fixed number of states is chosen from the list to continue simulation. Other states are deleted.

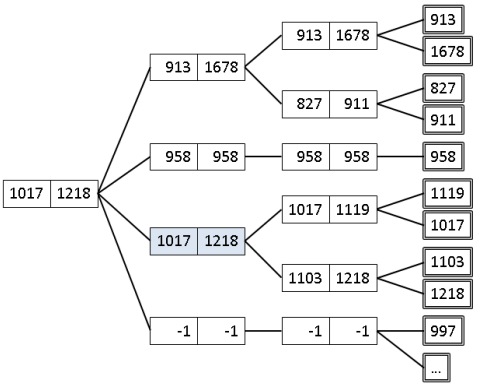

Fig. 3. The minimax algorithm. In the next step, the (1017. 1218) state will be selected as it maximizes the minimum value. The last state will not be fully explored as it contains a value lower that the highest already found minimum value.

5) Minimax Exploration: The Minimax exploration adapts the Minimax algorithm for the needs of the guided simulation. It is based on the following idea: At every step of the simulation, continue with the step, which seems to be the most promising. It explores all the subsequent states up to a certain depth (called *look ahead* and for each of the states evaluates the minimum and the maximum value of the metric for the subsequent states. Then it chooses the state with the maximal minimum value.

The implementation of the algorithm was optimized using alpha-beta pruning, which stops evaluating a branch when it finds a state whose minimum value is lower than the highest of already found values.

Fig. 4 shows an example of the minimax exploration.

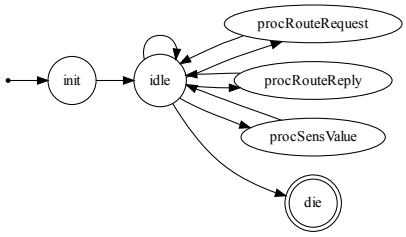

Fig. 4. Sensor node as a finite state automaton

IV. Model of a Wireless Sensor Network

For our experiments, we implemented a non-trivial operational model of a wireless sensor network with a routing protocol, application layer and a reputation-based security solution.

A. Network Model

The model consists of several nodes implemented as IF processes. Node number 0 represents the base station, the other nodes are regular sensor nodes. The network topology is set in the model as a graph where vertices are sensor nodes and edges represent pairs of nodes within radio range, which can communicate with each other.

When a node sends a message, it is delivered to all the nodes in its neighborhood (to all the nodes it is connected with). The receiving node analyzes the message and decides how to process it. Message delivery is reliable, messages are delivered to the FIFO queues of the receivers. The order in which several receiving nodes process a message, is arbitrary and is the cause of nondeterminism of the model.

Each node is equipped with a battery with a given initial capacity. Sending or receiving a broadcast message consumes an amount of energy set as a constant. Sending or receiving a unicast message also consumes a constant amount of energy, higher than in case of broadcast.

Sensor node is an IF process. Most of the time, the process is in *idle* state, waiting for input message. When it receives a message, it processes it and returns to the idle state. There are three more states which simplify the implementation of more complex operations. (See Fig. 4 for an abstract model of a sensor node.)

B. Network Operation

During the initialization one node is set as a sending node. This node will periodically send a value to the base station. In order to reach the base station in a multi-hop network, a routing protocol has to be used.

We implemented a simplified version of the well-known Ad hoc On-Demand Distance Vector (AODV) algorithm [9]. For each destination, each node stores the address of the next hop in the route (i.e. the address of the neighbor it will forward a message to). At the beginning, nodes have no information about the network topology.

When a node needs to find a route to the destination, it broadcasts a *route request* message. All nodes in its neighborhood receive the message. If a node knows how to reach the destination (it has an address of the next hop in its routing table), it sends a *route response* to the sender. If not, it stores the sender into its routing table and broadcasts the *route request*. This way the request is flooded throughout the network until it reaches its destination. The destination node answers with *route reply* message, which is forwarded via unicast messages back to the sender.

Nodes must ensure that they do not forward the same route request several times. Each route request is assigned a unique identifier and nodes store store the identifiers of the already forwarded messages. There is a customizable timeout after which the routing tables are erased and the new run of routing algorithm is executed. This way, the network can adapt to a loss of a node which ran out of energy.

This operational model will be in the future extended by introducing non-standard behavior of a node to simulate an attack, or by introducing more complex communication patterns.

C. Reputation System

To protect the integrity of the data transferred in the network, we implemented a reputation system based on the Watchdog mechanism [6]. It works in the following way: when a node is asked to forward a data packet, a neighboring node stores a copy of the packet in its memory and waits until it overhears the packet being forwarded. The neighbor checks whether the content of the forwarded packet is the same as of the original one. If it is different, it means that the node tried to change it. The node which noticed the misbehavior send an alarm message to all its neighbors. The neighbors delete all the routes going via the misbehaving node and put it on their blacklists. In the next run of the routing protocol, the blacklisted node is avoided.

D. End of Operation

When simulating lifetime of a network we set the following condition to end of operation (i.e. the goal state): the network stops operation when the first of the battery-equipped nodes runs out of energy.

E. Simulation Objective

The objective of the simulation is to find the worst case scenario with respect to the energy consumption. It means to find such a sequence of states in the space state that causes the network to stop operating in the shortest time.

In order to simplify the model, we did not include detailed simulation of time. The lifetime of the network has to be approximated by some other means - by the value of the timer that periodically requests the sending node to send data or by the number of messages sent (the former value being just a multiple of the latter) or by the number of messages delivered (giving more information about the behavior of the network when nodes are stopping working).

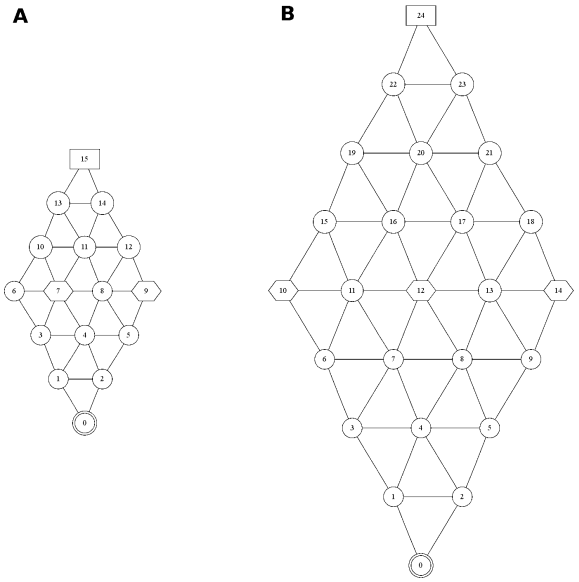

Fig. 5. The network topologies used. The rectangular node is the sending node, the double square node is the base station, the diamond node is the intruder.

We decided to use an *energy-per-message* metric. It is defined as:

$$epm = \frac{energyConsumed}{nbMsgDelivered}$$

where $energyConsumed$ is the total amount of energy consumed by the network to get to the state (a sum of energy spent by all battery-equipped nodes) and $nbMsgDelivered$ is the number of messages delivered to the base station.

The epm of the initial state is set as a value MAX, which is much higher than the epm of the consequent states.

F. Network Topologies

We used two networks for our experiments. In order to observe the communication in the reputation system, it was important that most of the nodes have at least three neighbors. We used a diamond shaped networks with 15 and 24 nodes, out of which 2, resp. 3 nodes are under the control of the intruder. (Fig. 5)

The intruder nodes behave in the following way: they participates actively on the routing protocol, i.e. they the routes to be established. However, when forwarding a data packet, they change its content (attack on the integrity). The neighboring nodes running the reputation systems should be able to detect this behavior and send alarm to other nodes to avoid routing via these nodes.

G. Battery Capacity

Setting different values for the capacity of node batteries is an efficient way to control the length and complexity of the simulation. We ran our experiments setting the battery capacity to 2000, 10000 and 50000 energy units.

Battery capacity: 2000	
Average epm:	1420.963
Minimum epm:	1122.167
Maximum epm:	1739.813
9th decile epm:	1652.235
Battery capacity: 10000	
Average epm:	835.656
Minimum epm:	727.705
Maximum epm:	937.274
9th decile epm:	914.605
Battery capacity: 50000	
Average epm:	629.394
Minimum epm:	556.875
Maximum epm:	693.846
9th decile epm:	690.570

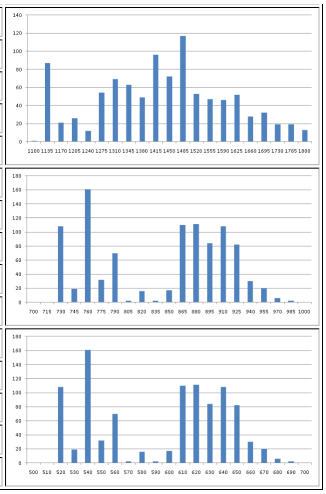

Fig. 6. Results and histogram of random executions of the network A.

Battery capacity: 2000	
Average epm:	2821.597
Minimum epm:	1593.000
Maximum epm:	12710.000
9th decile epm:	12015.500
Battery capacity: 10000	
Average epm:	3436.436
Minimum epm:	1061.556
Maximum epm:	29837.500
9th decile epm:	26164.500
Battery capacity: 50000	
Average epm:	9568.842
Minimum epm:	823.240
Maximum epm:	116467.500
9th decile epm:	96751.500

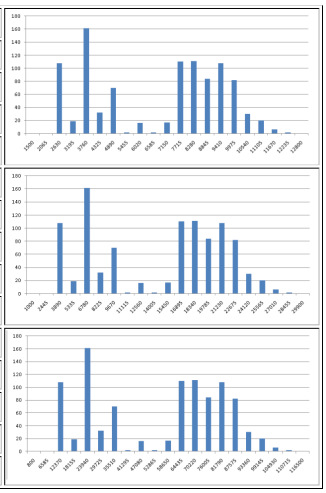

Fig. 7. Results and histogram of random executions of the network B.

V. Experimental Results

A. Random execution

We started the experimental part by running a set of random executions for each network topology and battery capacity. We executed each model 1000 times, created a histogram of the result values of the energy-per-message metric and calcualated the average, minimum and maximum value and the 9th decile.

For this analysis we omitted the lowest and the highest 5% of the results, which correspond to extreme values, which appeared as a consequence of the simplification of the implementation of the algorithms, which are not able to handle some extreme cases properly.

The results for the network A are shown in Fig. 6, for the network B in Fig. 7.

B. Guided Simulation

We tried all the algorithms with the three battery capacities on the network A. For the heap and onion heap exploration the maximum heap size was set to 512, the parallel exploration

Algoritm	epm	time (sec)	states
Battery capacity: 2000			
heap epm	1142.666667	454	5278045
onion heap epm	1558.000000	367	330331
parallel epm	1327.727273	38	1016787
minimax epm	1605.777778	10	204057
Battery capacity: 10000			
heap epm	732.640625	2525	30550366
onion heap epm	-	1427	-
parallel epm	752.570312	212	4669710
minimax epm	927.565217	52	1263945
Battery capacity: 50000			
heap epm	1557.806109	15649	185179616
onion heap epm	-	1356	-
parallel epm	-	434	-
minimax epm	691.445946	184	4698320

Fig. 8. Results of the guided simulation of the network A.

Algoritm	epm	time (sec)	states
Battery capacity: 2000			
minimax epm	12694.500000	202	533077
Battery capacity: 10000			
minimax epm	30015.500000	223	642114
Battery capacity: 50000			
minimax epm	117005.500000	377	1183254

Fig. 9. Results of the minimax exploration for the network B.

run 20 parallel executions and the minimax algorithm used the look ahead value of 3. The results are shown in Fig. 8.

We observed that for a high battery capacity, all algorithms except for the minimax algorithm either run out of memory (onion heap and parallel exploration) or take too long to finish (heap exploration). These algorithms also did not seem to provide useful results.

For the experiments with the bigger network B we therefore decided to proceed only with the minimax algorithm. The results are shown in Fig. 9.

VI. Analysis

The results confirm the hypothesis stated in [8]. The greedy algorithm (heap exploration) gets stuck in the local optima and the sequence it finds is under average. This strategy is not usable for finding a maximum execution sequence.

The extension with several heaps and a combination of greedy and random approach (onion heap exploration) seems to provide slightly better result for a small battery capacity (i.e. when a state space is quite small). However, the result is still under average and it does not seems to solve the local optima problem. We were not able to run the algorithm on a bigger state space, due to the limitation of the implementation.

The parallel exploration seems to be more successful when the battery capacity is low, but still the result for a medium capacity is under average.

On the contrary, the minimax strategy seems to provide a meaningful results in all the cases. For the low battery capacity the result is above average. For the medium and high battery capacity the results obtained is higher than the 9th decile, i.e. it is in the highest 10% of the results in the histogram.

For the network B, the results of the minimax strategy are even better - all the results are in the top 10%.

Comparing the time complexity, the minimax algorithm performs significantly better than all the other algorithms.

VII. Conclusion

We explored the feasibility of guided simulation for finding execution sequences with a particular characteristic in a simulation of a wireless sensor network. We experimented with those recently proposed strategies, which seemed to be adaptable for our purposes.

We confirmed that the greedy strategy does not perform very well as it tends to get stuck in a local optima and requires too much of time and energy resources. The optimizations with the onion and parallel exploration did not seem to bring significant improvements.

We proposed to use a strategy based on the Minimax algorithm and from our experimental results it seems to be a right choice for the purpose. The algorithm is able to find a good execution sequence (one of the best 10% of the possible sequences) in a reasonable time.

In the future, we think it might be interesting to implement the guided simulation as an extension of a real-world simulation tool and experiment with its usefulness for the network simulation and the development of new algorithms.

References

[1] The network simulator - ns-2, http://www.isi.edu/nsnam/ns/

[2] OMNeT++, http://omnetpp.org/

[3] Bozga, M., Graf, S., Ober, I., Ober, I., Sifakis, J.: The IF toolset. In: Formal Methods for the Design of Real-Time Systems, pp. 237–267 (2004), http://www.springerlink.com/content/qqtpa45muh6c9kf1

[4] Dill, D.L.: What's between simulation and formal verification? (extended abstract). In: Proceedings of the 35th annual Design Automation Conference. pp. 328–329. ACM, San Francisco, California, United States (1998), http://portal.acm.org/citation.cfm?id=277138

[5] Ho, P.H., Shiple, T., Harer, K., Kukula, J., Damiano, R., Bertacco, V., Taylor, J., Long, J.: Smart simulation using collaborative formal and simulation engines. In: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design. pp. 120–126. IEEE Press, San Jose, California (2000), http://portal.acm.org/citation.cfm?id=602931

[6] Marti, S., Giuli, T.J., Lai, K., Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: Proceedings of the 6th annual international conference on Mobile computing and networking. pp. 255–265. ACM, Boston, Massachusetts, United States (2000), http://portal.acm.org/citation.cfm?id=345955

[7] Mounier, L., Samper, L., Znaidi, W.: Worst-case lifetime computation of a wireless sensor network by model-checking. In: PE-WASUN '07: Proceedings of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor,and ubiquitous networks. pp. 1–8. ACM, New York, NY, USA (2007)

[8] Paula, F.M.D., Hu, A.J.: An effective guidance strategy for abstraction-guided simulation. In: Proceedings of the 44th annual Design Automation Conference. pp. 63–68. ACM, San Diego, California (2007), http://portal.acm.org/citation.cfm?id=1278498

[9] Royer, E.M., Das, S.R., Belding-Royer, E.M., of Cincinnati, U., Perkins, C.E.: Ad hoc On-Demand distance vector (AODV) routing. http://tools.ietf.org/html/rfc3561 (Jul 2003), http://tools.ietf.org/html/rfc3561

[10] Shyam, S., Bertacco, V.: Distance-Guided hybrid verification with GUIDO. In: Design, Automation and Test in Europe, 2006. DATE '06. Proceedings. vol. 1, pp. 1–6 (2006), 10.1109/DATE.2006.244050

[11] Yang, C.H., Dill, D.L.: Validation with guided search of the state space. In: Proceedings of the 35th annual Design Automation Conference. pp. 599–604. ACM, San Francisco, California, United States (1998), http://portal.acm.org/citation.cfm?id=277201