J-Sim: An Integrated Environment for Simulation and Model Checking of Network Protocols

Ahmed Sobeih, Mahesh Viswanathan, Darko Marinov and Jennifer C. Hou

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA

Presented by: Ahmed Sobeih

NSFNGS 2007 Workshop, held in conjunction with IPDPS 2007
Long Beach, California, USA, March 25-26 2007

Problem Definition: What?

- Network simulators (e.g., ns-2, J-Sim)
  - Build a simulation model of a network protocol
  - Evaluate its performance in scenarios provided by the user

- Deficiency of traditional network simulators:
  - Only evaluate performance in scenarios provided by the user, but can *not* exhaustively analyze possible scenarios for “correctness” of either the simulation model or the protocol itself.

  - Examples of correctness (protocol-level requirements):
    - Can a routing protocol suffer from routing loops?
    - Can an attacker break a security protocol?

Extend network simulators with verification capabilities

Motivation: Why?

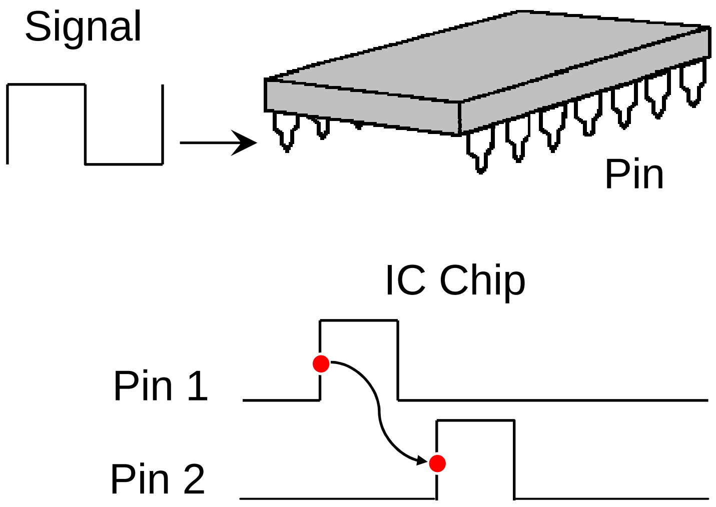
- Network simulators have been widely used for decades
- The earlier an error is found the better
  - Errors in the simulation model may lead to incorrect experimental results
  - Errors in the network protocol itself may happen after deployment
- Building another model specifically for verification purposes is time- and effort-consuming and error-prone
  - Network protocol designers are more familiar with network simulators written in imperative languages (e.g., C++, Java)
- Translating programming languages (e.g., Java) into the input modeling languages of conventional model checkers
  - May not be always feasible. Requires that each language feature of Java have a corresponding one in the destination modeling language.
  - Making use of the simulation model that the protocol designer has to build anyway for performance evaluation purposes

Can we have <u>a single integrated</u> tool providing *both* Performance Evaluation *and* Verification?

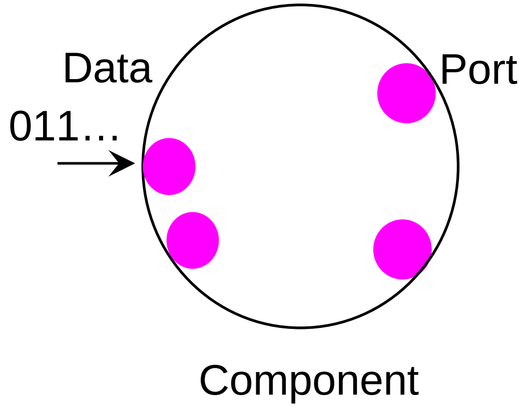
J-Sim (http://www.j-sim.org)

- **Autonomous Component Architecture (ACA)**
  - a component-based software architecture

- **ACA closely mimics the Integrated Circuit (IC) design**



At design time, an IC is bound to a certain specification in the databook, instead of being bound to ICs that interact with it.

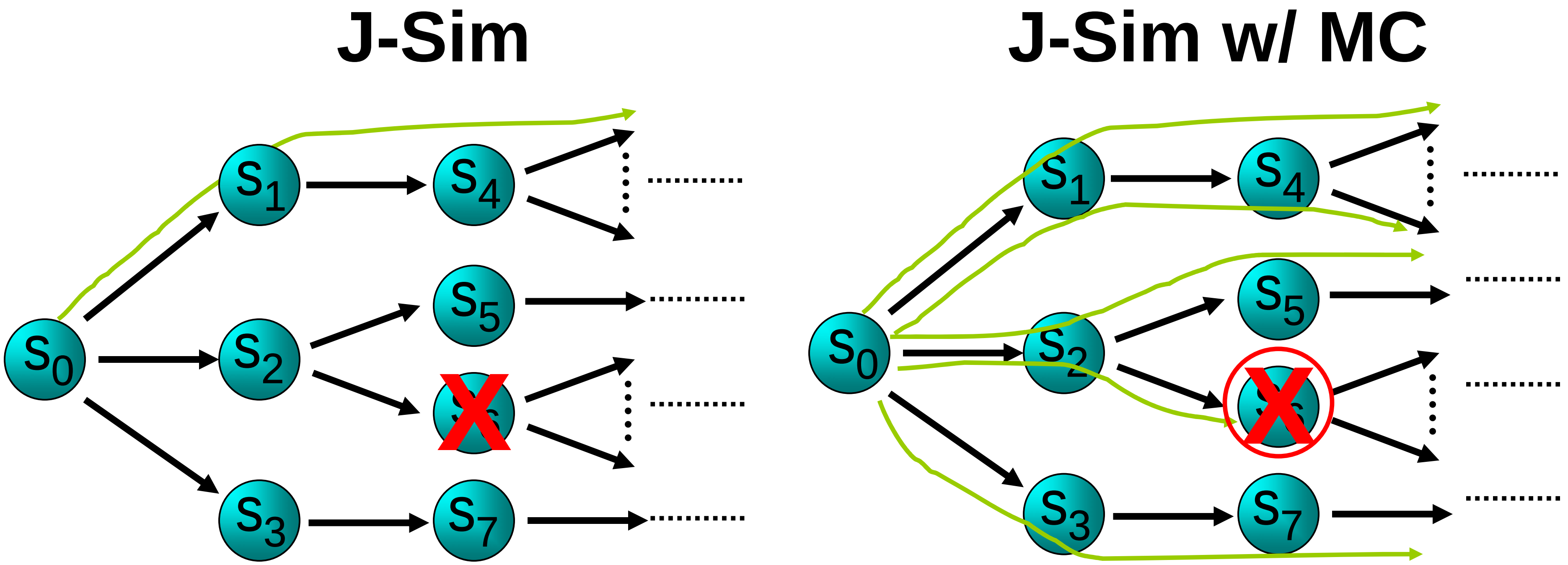


At design time, a component is bound to a certain contract, instead of being bound to components that interact with it.

ACA realizes the notion of a "software" IC

Model Checking Framework in J-Sim

- Stateful on-the-fly explicit-state model checking in J-Sim
  - Explore the state space created by the simulation model of a network protocol up to a (configurable) maximum depth of transitions
  - No changes to the core design and implementation of J-Sim



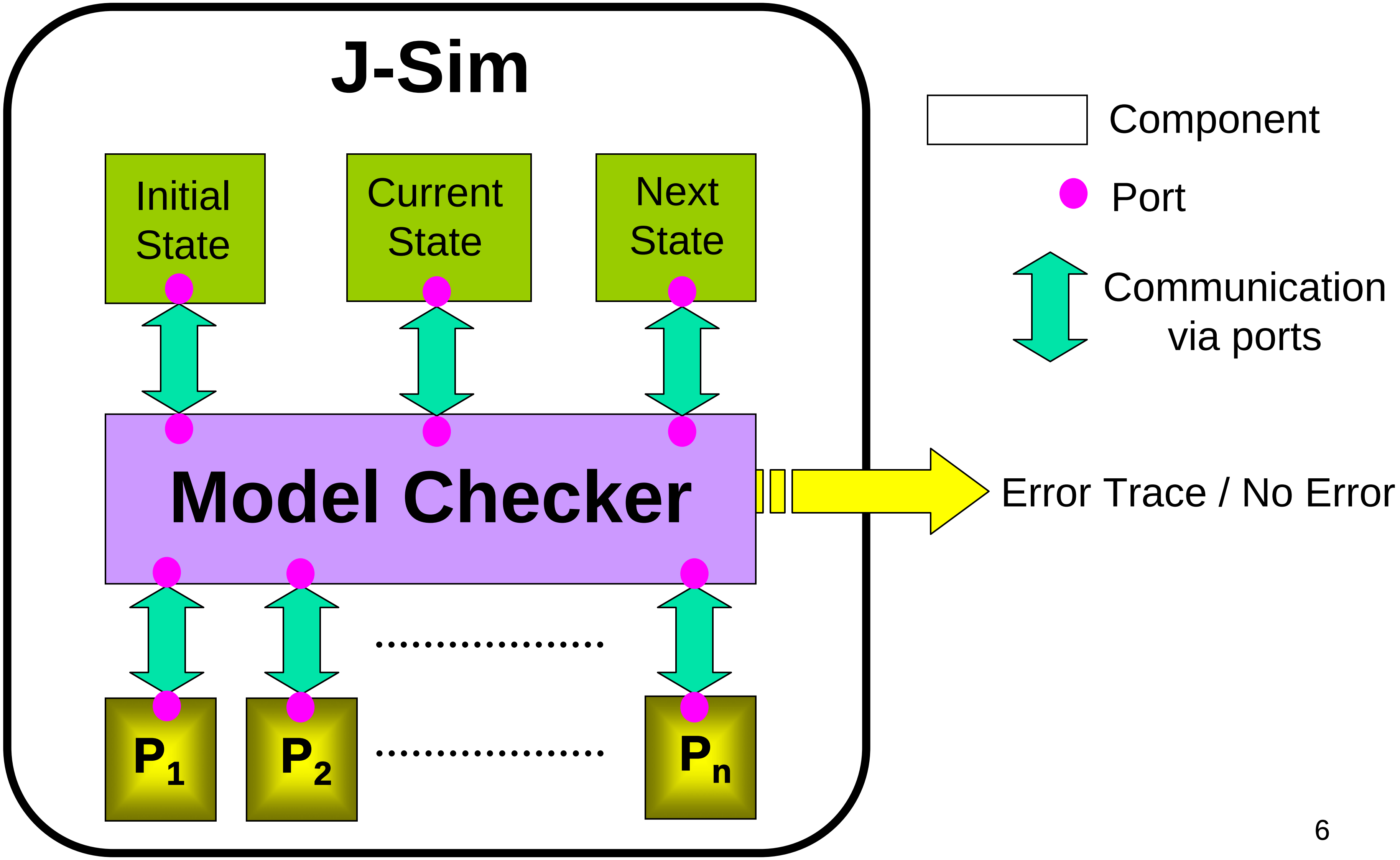
J-Sim

J-Sim w/ MC

X *denotes a violation of a safety property*

Model Checking Framework in J-Sim (cont'd)

- Build the model checker as a *component* in the ACA of J-Sim

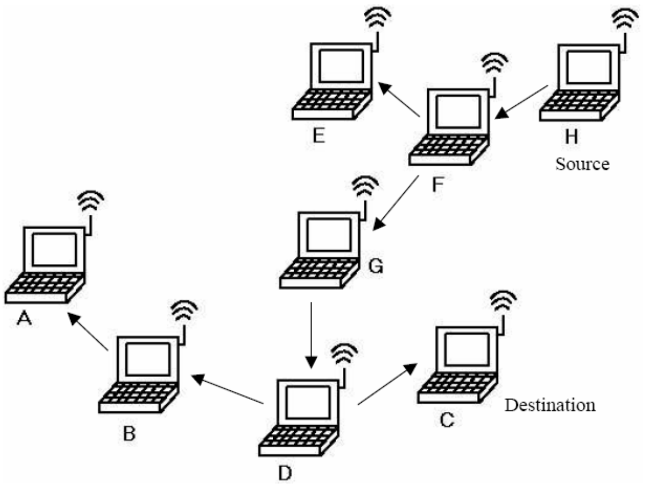


Evaluation and Results

- AODV routing for MANETs
- Reasonably complex network protocol
  - 1200 LOC (excluding the J-Sim library)
- Representative routing protocol for MANETs
- Safety property
  - Loop-free property of routing paths
- Infinite state space
- Handling state space explosion:
  - Making use of protocol-specific heuristics to develop best-first search (BeFS) strategies
  - Exploit properties inherent to the *network protocol* and the *safety property* being checked

AODV Case Study

- Routing protocol: build and maintain routing table entries (RTEs)
- In AODV, the RTE at node *n* for a destination *d* contains the following fields: $nexthop_{n,d}$, $hops_{n,d}$, $seqno_{n,d}$
- On route timeout: invalidate (but not delete) RTE, increment $seqno_{n,d}$, $hops_{n,d} \leftarrow$ infinity



- Loop-free property:
  - A node can *not* occur at two points on a path
  - Consider two nodes *n* and *m* such that $nexthop_{n,d} = m$

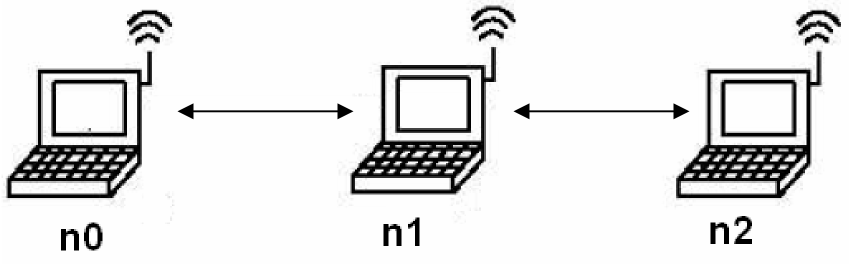
$$((seqno_{n,d} < seqno_{m,d}) \vee (seqno_{n,d} == seqno_{m,d} \wedge hops_{n,d} > hops_{m,d}))$$

- AODV-BeFS: considers a state s_1 better than a state s_2 if the ***number of valid RTEs to any node*** in s_1 is greater than that in s_2.

AODV Case Study (cont'd)

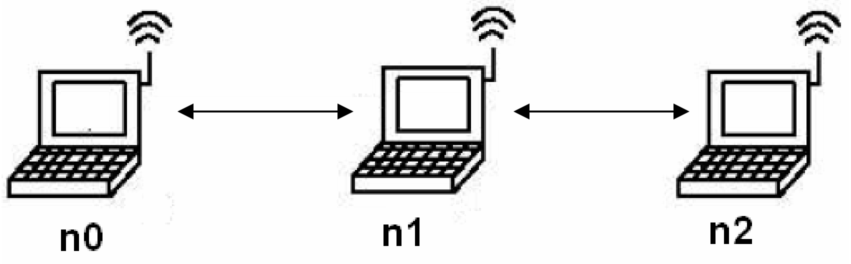
Errors discovered and injected

- CE1: An error in the J-Sim simulation model of AODV caused by not following part of the AODV specification when an AODV process restarts



$nexthop_{n0,n2} = n1$ $seq_{n0,n2} > seq_{n1,n2}$

- Two manually injected, but subtle, errors:
  - CE2: Not to increment $seqno_{n,d}$ when an RTE is invalidated
  - CE3: Deleting (instead of invalidating) the RTE

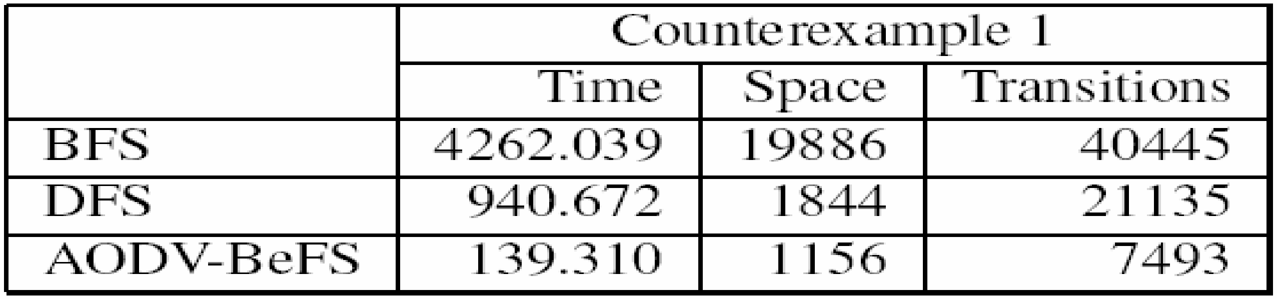


$nexthop_{n0,n2} = n1$ $nexthop_{n1,n2} = n0$

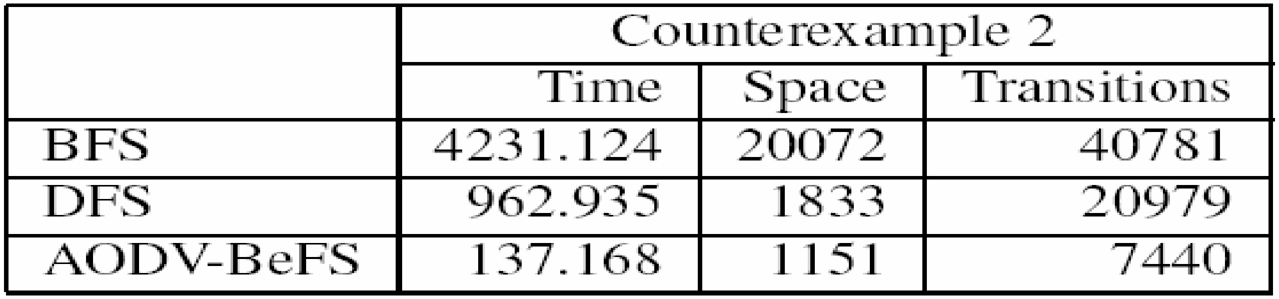
AODV Case Study (cont'd)

Performance of the search strategies

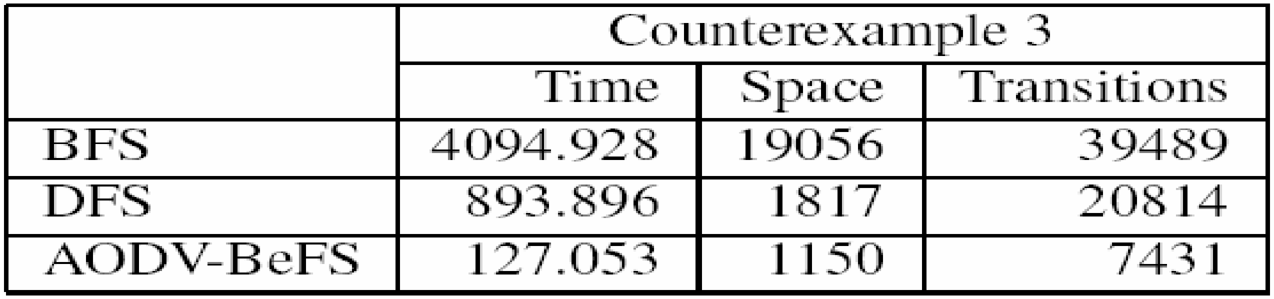
Time (in seconds) and space (in number of states explored) requirements and the number of transitions executed for finding the three counterexamples in a 3-node chain ad-hoc network using different search strategies. MAX_DEPTH = 10



	Counterexample 1		
	Time	Space	Transitions
BFS	4262.039	19886	40445
DFS	940.672	1844	21135
AODV-BeFS	139.310	1156	7493



	Counterexample 2		
	Time	Space	Transitions
BFS	4231.124	20072	40781
DFS	962.935	1833	20979
AODV-BeFS	137.168	1151	7440



	Counterexample 3		
	Time	Space	Transitions
BFS	4094.928	19056	39489
DFS	893.896	1817	20814
AODV-BeFS	127.053	1150	7431

Conclusion

- Extending J-Sim (www.j-sim.org) with verification capabilities
- Several case studies of fairly complex network protocols
  - ARQ, AODV for MANETs, Directed Diffusion for WSNs
  - The framework is general enough and not tied to a particular network protocol
  - The framework can handle larger network topologies
- A methodology for the model checking of another network protocol
- Making use of protocol-specific heuristics to develop best-first search (BeFS) strategies
  - Using analogies between AODV and directed diffusion
  - Recommend exploiting properties inherent to the *network protocol* and the *safety property* being checked

Future Work

- **Comparison with Java PathFinder (JPF), a model checker for Java programs**
  - Use JPF to model-check the network protocols in J-Sim
  - Compare the model checking framework in J-Sim with that of JPF
  - Assess the pros and cons of building a model checker in J-Sim instead of using an existing model checker for Java programs such as JPF

- **Class-specific (instead of protocol-specific) heuristics**
  - Devise efficient heuristics for each class of protocols (e.g., routing, coverage and connectivity, localization, etc.)
  - If a network protocol belonging to a certain class is to be model-checked, the user can use the appropriate heuristic for that class instead of having to start from scratch