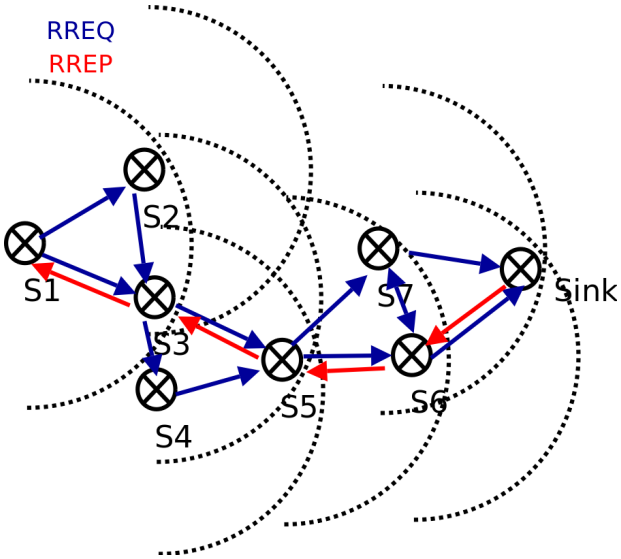
Norsk Regnesentral
NORWEGIAN COMPUTING CENTER

Report

Validation of *Creol* Models for Routing Algorithms in Wireless Sensor Networks

CREDO



Report no 1024
Authors Wolfgang Leister
Joakim Bjørk
Rudolf Schlatte
Andreas Griesmayer
Date February 10th, 2010
ISBN 978-82-539-0534-1

Norsk Regnesentral
NORWEGIAN COMPUTING CENTER

The authors

Wolfgang Leister, Chief Research Scientist at Norsk Regnesentral, received the Dr.rer.nat. degree in 1991 from the Universität Karlsruhe, Germany. His research interests cover multimedia, computer graphics, computer and sensor networks, health care applications, mobile systems, and free software. Recent projects include ADIMUS (Adaptive Internet Multimedia Streaming), DIGEKS (ICT-Based exams), SAMPOS (Strategies for Seamless Deployment of Mobile Patient Monitoring Systems), and CREDO (Modeling and Analysis of Evolutionary Structures for Distributed Services).

Joakim Bjørk is a PhD student at the University of Oslo. He received his cand. scient. degree in 2006. His research interests cover modelling and analysis of concurrent systems.

Rudi Schlatte is researcher at the University of Oslo, after recently having finished his PhD studies at Graz University of Technology. He received a master degree in Telematics in 2001, and worked as researcher at Joanneum Research, Austria. His research interests include modelling and testing of real-time distributed systems, and design of modelling languages. During his studies, he spent two years as a Research Fellow at the United Nations University International Institute for Software Technology in Macao, China.

Andreas Griesmayer was post doctoral fellow at UNU-IIST Macao until October 2009, at which time he moved to France for a position as post doctoral researcher at Verimag, Grenoble. He received his Dr. tech in Computer Science in 2007 from the Technical University Graz, Austria. His research interests cover verification, fault detection and repair of software, and related techniques as modelling, abstraction, and model checking. Before CREDO, he worked on the EU project PROSYD (Property Based System Design).

Norwegian Computing Center

Norsk Regnesentral (Norwegian Computing Center, NR) is a private, independent, non-profit foundation established in 1952. NR carries out contract research and development projects in the areas of information and communication technology and applied statistical modeling. The clients are a broad range of industrial, commercial and public service organizations in the national as well as the international market. Our scientific and technical capabilities are further developed in co-operation with The Research Council of Norway and key customers. The results of our projects may take the form of reports, software, prototypes, and short courses. A proof of the confidence and appreciation our clients have for us is given by the fact that most of our new contracts are signed with previous customers.

Norsk Regnesentral
Norwegian Computing Center
Postboks 114, Blindern
NO-0314 Oslo, Norway

Besøksadresse
Office address
Gaustadalléen 23
NO-0373 Oslo, Norway

Telefon · telephone
(+47) 22 85 25 00
Telefaks · telefax
(+47) 22 69 76 60

Internett · internet
www.nr.no
E-post · e-mail
nr@nr.no

Title	**Validation of *Creol* Models for Routing Algorithms in Wireless Sensor Networks**
Authors	**Wolfgang Leister, Joakim Bjørk, Rudolf Schlatte, Andreas Griesmayer**
Quality assurance	Bjarte M. Østvold, Trenton Schulz
Date	February 10th, 2010
ISBN	978-82-539-0534-1
Publication number	1024

Abstract

We validate a *Creol* model of AODV by evaluating functional properties using simulation and component testing. We define a structure for evaluating and validating models of wireless sensor networks, which we intend to use as a reference for future evaluations of algorithms. We use the categories of techniques, perspectives, arrangements and properties to structure the evaluation work. We show, by way of the AODV algorithm, how network simulations and component testing can be employed to evaluate a large list of properties. We also show which properties are most suited to be evaluated by which technique, perspective, and arrangement.

Keywords	*Creol*, model checking, simulation, component testing, evaluation of properties
Target group	Researcher Community
Availability	Open
Project	CREDO
Project number	320362
Research field	Formal Methods, Sensor Networks
Number of pages	23

Contents

1 Introduction

Wireless sensor networks (WSN) (Akyildiz et al., 2002) are used in an increasing number of applications. These networks consist of spatially distributed autonomous sensor nodes that communicate wirelessly. Each sensor node in a WSN is an electronic device that contains components to perform the tasks of sensing, processing, sending and receiving of sensor data.

From a technical point of view, these sensors form a network, in which data packets from several source nodes are forwarded towards sink nodes, possibly via other nodes that serve as a forwarding device. In each node forwarding and routing[1] (Akkaya and Younis, 2005; Al-Karaki and Kamal, 2004) algorithms are employed in the transportation of the sensor data from the sender to the sink node.

In this paper we present models of the *Ad hoc On Demand Distance Vector* (AODV) routing algorithm (Perkins et al., 2003), where different properties are represented and evaluated. AODV has been designed especially for wireless ad-hoc networks (Chakeres and Belding-Royer, 2004; Gomez et al., 2006). Proving that forwarding and routing algorithms work as intended is a complex task due to the autonomous behaviour of the sensor nodes in a WSN. Data processed by a WSN must fulfil properties such as quality of service (QoS), timing, delay, network throughput, packet delivery ratio, network connectivity, energy consumption, mobility, and resource consumption.

In order to evaluate these properties, we model the components of a WSN and employ simulation, testing, and model checking techniques. The purpose of the present paper is to apply the model checking methods of *Creol* rather than developing new routing or forwarding algorithms. To give an example on how to use *Creol*, we use well-known algorithms that have been evaluated previously.

The contribution of this paper is to structure the work around evaluating and validating *Creol* models of wireless sensor networks. We intend to use this structure as a reference for future evaluations of algorithms used for wireless sensor networks. We use the categories of techniques, perspectives, arrangements and properties to structure the evaluation work. We show, by way of the AODV algorithm, how network simulations and component testing can be employed to evaluate a large list of properties. We also show which properties are best suited to be evaluated by technique, perspective, and arrangement.

The remainder of this paper is organised as follows: After shortly presenting *Creol* models, related work, and the AODV model developed previously (Leister et al., 2009) we explain techniques, perspectives, arrangements, and properties used in the validation process (Section 2). Results from network simulation (Section 3) and component testing (Section 4) are presented before concluding this document (Section 5).

1. Routing is the process of selecting paths in a network along which to send network traffic, while forwarding is used to transport the data.

1.1 Executable *Creol* Models

Creol (Johnsen and Owe, 2007; Kyas, 2009) is an *object-oriented* modelling language that provides an abstract, executable model of the implementation of components. The *Creol* tools are part of the *Credo* tool suite (Grabe et al., 2009) that unifies several simulation and model checking tools. The *Credo* tools support integrated modelling of different aspects of highly re-configurable distributed systems, both structural changes of a network, and changes in the components. The *Credo* tools offer formalisms, languages, and tools to describe properties of the model in different levels of detail; these formalisms include various types of automata, procedural, and object-oriented approaches.

To model components, *Creol* provides behavioural interfaces to specify inter-component communication. We use intra-component interfaces together with the behavioural interfaces to derive test specifications to check for conformance between the behavioural model and the *Creol* implementation.

In *Creol*, types are separated from classes, and (behavioural) interfaces are used to type objects. Objects are concurrent, i.e., conceptually, each object encapsulates its own processor. *Creol* objects can have active behaviour. During object creation a designated run method is automatically invoked. *Creol* allows flexible object interaction based on asynchronous method calls, explicit synchronisation points, processor release, and underspecified (i.e., nondeterministic) local scheduling of the processes within an object. *Creol* supports software evolution by means of runtime class updates (Yu et al., 2006). This allows for runtime configuration of the components in a distributed manner.

Creol (Kyas, 2009) includes a compiler and type-checker, a simulation platform based on Maude (Clavel et al., 2001), which allows simulation, guided simulation, model testing, and model checking. As a result of our modelling efforts, the extension *CreolE* (Leister, 2009) has been proposed.

1.2 Related Work

Showing functional correctness and non-functional properties for algorithms employed for WSN helps the developers in their technical choices. Developers use a variety of tools, including measurements on real implementations, simulation, and model-checking. When developing algorithms for packet forwarding in a WSN, simulation results must be compared with the behaviour of known algorithms are necessary to get a result approved (Stojmenovic, 2008).

There are several approaches using simulation, testing, and model checking during the development process, using one or more of the following: (a) modelling; (b) traces; (c) runtime monitoring by integrating checking software into the code (instrumentation) (Musuvathi et al., 2002); or (d) generating software from models automatically (Mozumdar et al., 2008).

Simulation systems are used to analyse performance parameters of communication networks, such as latency, packet loss rate, network throughput, and other metrics. Most of these systems use discrete event simulation. Examples for such simulation systems

include Opnet[2], OMNeT++[3], and ns-2[4], or mathematical frameworks like MathWorks[5]. Most of these tools have specialised libraries for certain properties, hardware, and network types.

The CMC model checker (Musuvathi et al., 2002) has been applied on existing implementations of AODV, by checking an invariant expressing the loop-freeness property. In that work both specification and implementation errors were found, and were corrected in recent versions of the specification and implementation. CMC interfaces C-programs directly by replacing procedure calls with model-checker code, thus avoiding the need to model an algorithm.

The model checking tools SPIN and *UPPAAL* have been used to verify properties for the correct operation of ad hoc routing protocols (Wibling et al., 2004). They use *Propagation Localized Broadcasting with Dampening* (PLBD) as a basic operation, and perform model checking on the LUNAR and DSR algorithms (Wibling et al., 2005) in *UPPAAL*. Both LUNAR and DSR are related to AODV, but use different mechanisms. A timing analysis in *UPPAAL* uncovered that many AODV connections unnecessarily timed out before a route could be established in large networks (Chiyangwa and Kwiatkowska, 2005). Timed automata implemented in *UPPAAL* has been used for validating and tuning of temporal configuration parameters and QoS requirements (Tschirner et al., 2008) in network models that allow dynamic re-configurations of the network topology.

Distributed applications can be described in terms of components interacting in an open environment. These components are used in an object-oriented framework based on the mechanisms of *Creol* (Johnsen et al., 2007). This framework models components and the communication between these components, and executes the models in rewriting logic. Different communication patterns, communication properties, and a notion of time are supported. We modelled the lower communication layers using tight, loose, and wireless links.

The OGDC-algorithm used in certain sensor networks has been simulated and model-checked in Real-Time Maude (Ölveczky and Thorvaldsen, 2007). The simulation results in Real-Time Maude have been compared with simulation results in ns-2, which uncovered weaknesses in the concrete ns-2 simulation.

1.3 Modelling the Components and the Routing Algorithm

We base our work on a previously defined model of AODV in a WSN in *Creol* (Leister et al., 2009) that expresses each node and the network as objects with an inner behaviour. The interfaces of the objects describe the communication between the nodes and the network; i.e., the nodes do not communicate directly with each other. In Figure 1, we show the object structure of the model, including the most important interfaces of a node.

Within the nodes, the behaviour of AODV is implemented as routines without explicitely modelling the internal object structure. The model of AODV in *Creol* is similar

2. See http://www.opnet.com/.
3. see http://www.omnetpp.org/.
4. See http://nsnam.isi.edu/nsnam/.
5. The MathWorks – MATLAB and Simulink for Technical Computing, www.mathworks.com. The MathWorks is a combination of Simulink and Matlab.

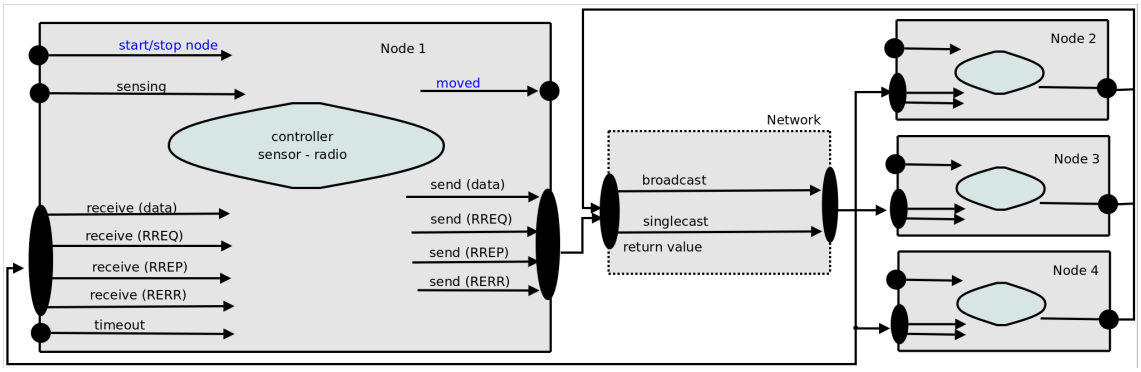


Figure 1. Objects of a WSN model and their communication interfaces.

to implementations in the real world. The purpose of a routing algorithm is to establish a path between a source node and a sink node, so that data can flow from the source node to the sink node. AODV is a reactive routing protocol that builds up the entries in the dynamic routing tables of nodes only if needed. AODV can handle the network dynamics, e.g., varying wireless link qualities, packet losses, and changing network topologies.

When a node wants to send a message to a sink node whose next hop cannot be retrieved from the routing table the node initiates a route discovery procedure by broadcasting RREQ (route request) messages. Nodes that receive a RREQ message will either send a RREP (route reply) message to the node which originated the RREQ message if the route is known; otherwise the present node will re-broadcast the RREQ message. This procedure continues until the RREQ message reaches a node that has a valid route to the destination node. The RREP message is unicast to the source node through multi-hop communications; as the RREP message propagates, all the intermediate nodes establish routes to the destination. After the source node has received the RREP message, a route to the destination has been established, and data packages can be sent along this route.

The essential entries of the routing table include the next hop, a sequence number, and the hop count to the sink node. The latter is the most common metric for routing to choose between routes when multiple routes exist. The sequence number is a measure of the freshness of a route.

When communication failures imply a broken route, the node that is unable to forward a message will inform other nodes, so that the routing tables can be updated. To do this it sends a RERR (route error) message along the reverse route that is also stored in the nodes. Thus the source node will become aware of the broken route, and initiate a new route discovery procedure.

2 Simulation, Component Testing, Model Checking

In the following section we show how to evaluate and validate the functional behaviour of the AODV model using *Creol* and the *Credo* methodology (Grabe et al., 2009). We present the *techniques*, *perspectives*, *arrangements*, and *properties* necessary for the valida-

tion. The results of the evaluation are compared to results from the real world, and from other evaluations of AODV. We also show how to evaluate selected non-functional properties.

2.1 Techniques for Simulation, Testing, and Model Checking

In order to evaluate the properties of a model, several *techniques* are used to provide the necessary technical measures and procedures to make a model amenable to verification. In general, the following modifications can be applied to the model in preparation for simulation, testing, and model-checking:

Auxiliary variables: Additional variables are added to the model to improve the visibility of a model's behaviour. These auxiliary variables must not alter the behaviour and are implemented as counter variables, lists of objects, etc. to each object in the model. Their content is updated when certain relevant events happen (e.g., a counter is incremented when a new instance is created). When running a simulation step-by-step these values can be excerpted from the state information, and visualised[6]. In other cases we inspect the result from the state information when the *Creol* program stops.

Assertions: Depending on the functional requirements, it might be necessary to add assertions in the code. While a number of properties can be checked after at the final state, with the help of the aforementioned auxiliary variables, properties on the transient behaviour of the model require a check during runtime. For such cases, *Creol* provides *assertions* that stop the execution of a model when the condition is violated. The state that caused the violation of the property is then shown for further analysis.[7]

Monitors: Properties that go beyond simple assertions require the use of *monitors*, pieces of software that run in parallel to the actual model. A monitor follows the behaviour of the model and maintains a state to decide the validity of a path. It therefore constitutes an automaton. (For example: *Acknowledgements may only be sent if there was a request previously*).

Guarded execution: To be able to check the behaviour of the model under different conditions, while still maintaining reproducibility of the runs, nondeterministic decisions are replaced by calls to a guarding object, the *DeuxExMachina*. This technique also specifies certain parameters of the environment, like failure rates of the network.

Fault injection: Error recovery properties from error states are checked by adding a misbehaving node, possibly after a certain time. E.g., switch off node when energy is used up, or inject other errors. This can be implemented by sub-classing nodes, and implementing certain misbehaving routines in the subclass.

---

6. In some systems a debugging tool can be used. Since *Creol* does not provide such a tool we do not consider the use of a debugger in this context.

7. Examples: *check packet delivery ratio at final states; are there destination nodes where the number of packets is below a certain threshold; check number of hops or time-steps in timed model for all messages arrived at the sink node.*

Property search: Apply search commands for certain conditions that can be specified, in order to check whether these conditions hold for all or certain states applying specific search strategies. Currently, property search must be specified in *Maude* for a *Creol* model.

An instrumented *Creol* model can be used for the different verification and testing techniques in the *Credo* methodology: symbolic simulation, guarded test case execution, and model checking using the search capability of the underlying *Maude* system. Currently, the auxiliary variables for assertions and the state of the monitors need to be added as *Creol* code that is executed together with the model code. This increases the size of the states and therefore poses a handicap for model checking. The use of *Creol functions* that can be excluded from the state in model checking would be desirable. This feature is, however, not yet available in the current tools.

2.2 Perspectives

A *perspective* describes the scope of an evaluation. For the AODV model we developed two perspectives: (a) observing the behaviour of the entire network configuration including all nodes and the network; (b) observing the behaviour of one node.

Testing, simulation, and model checking can be performed from different perspectives and levels of detail for a given model. For the AODV algorithm, a holistic perspective focuses on the networking aspect of the nodes, implementing all the involved nodes and the network in one model. However, for model checking such a model leads to a high number of states, and long execution time. Therefore, for realistic models the networking perspective becomes unfeasible.

For the perspective of testing a single node we suggest a different way: We use the same model code for the nodes as in the holistic perspective, but instantiate only one node explicitly. The network is replaced by a *test driver* object that impersonates the network and the remaining nodes. The behaviour and responses of the test driver are determined by a rule set that is derived from traced messages between the nodes, e.g., from a real-world case or calculated from a different model.

We have implemented both modelling and testing from the holistic (networking) perspective, and from a node-perspective (component-testing). Section 4 describes component-testing in detail.

2.3 Arrangements

An *arrangement* denotes a set of configuration settings that influences of how the model operates. As examples we mention the use of untimed or timed models, possible changes of the node topology, perfect or unreliable communication, communication failures, timeouts, energy consumption, etc. The single arrangement entities can be switched on in the *Creol* model. Depending on the scenario to be simulated, tested or model-checked, the arrangement is chosen.

2.3.1 Communication behaviour

In our model, the communication behaviour can be set to be either reliable, non-deterministic, or one of several packet loss patterns. Note that non-deterministic in a simulation will not be useful, since the underlying *Maude* interpreter will always choose the same value due to implementation-specific reasons. Using the different communication behaviour, we can study how the algorithm behaves when communication packets can get lost. The communication patterns can be set for both broadcast and singlecast communication patterns.

2.3.2 Topology changes

We can model topology changes to check the robustness of the protocol. These topology changes can be triggered by certain events, e.g., after a certain amount of messages, or after a certain amount of time intervals using a timed model. A topology change affects the connection matrix in the network, and will trigger the AODV algorithm to find new routes following communication failures in the model.

Note that for model checking, the topology needs to be re-installed for a state when another branch is searched. Therefore, the use of a timed model combined with topology changes is most viable.

2.3.3 Timed model

The model time is divided into discrete time steps, and the entire model (i.e., all nodes and the network) is synchronised to be within the same time interval at any time. Using the timed model we can, for instance, reason about messages being sent simultaneously, which eventually will lead to packet loss. The idea in the present model is to have at most one action per time-step. The synchronisation is performed using a global clock in the network object, and an internal clock in each node which are synchronised as soon as at most one task is performed in one or more nodes.

Using the timed model, the effect of collisions can be shown, without using non-deterministic packet loss.

2.3.4 Energy consumption

Reasoning about energy consumption can be made by giving each sensor node a certain amount of energy. For each operation performed on a specific node we subtract a certain amount of energy until the capacity of this node is too low to perform operations on the radio, thus indicating a malfunction of the sensor node. In our model, a sensor node with too little energy does not perform any actions. In principle, malfunctioning nodes indicate a topology change of the network, since given paths are no longer valid.

Using the energy-consumption arrangement we can identify in which cases an energy-restricted network can perform communication, for how many messages, and for how long (in the case of a timed model). We can show whether the algorithm can work given certain energy capacities for each sensor node; or whether nodes get empty before all messages are sent, and whether the network can find routes around an energy-empty node.

Note, that arrangements to reason about memory and buffer sizes can be implemented similarly. However, when maximum memory size is reached, a node will temporarily stop to perform certain actions.

2.3.5 Timeout

Communication and routing algorithms employ timeouts in order to work in environments where errors can occur; e.g., messages are sent repeatedly in case an expected reply has not been received from the network. In the untimed arrangement timeouts can potentially be processed in the time-interval after a message is sent until a corresponding answer is received. We use a global object (the DeusExMachina object) to decide whether at a given point in time a timeout will occur. When a timeout occurs, a given message will be resent.

2.4 Functional Properties

A *functional property* is a concrete condition that can be checked for given arrangements. For AODV, we show the following functional properties: (a) correct-operation, (b) loop-freeness, (c) single-sensor challenge-response properties, (d) shortest-path, (e) deadlock-freeness (both, for node, and for protocol), (f) and miscellaneous composed system properties.

2.4.1 The correct-operation property (a)

As defined by Wiebling et al. (Wibling et al., 2004, 2005) a path *exists* between two nodes when a *path is valid* for some duration longer than what is required to set up a route from sender to receiver. For AODV, the number of time-steps needed is minimum twice the number of the nodes in the shortest route. After a route is set up, it can be used by payload packets until this route becomes invalid.

For a routing algorithm to be correct, it must find a path given that a path exists. To check whether a model fulfils this property, we need to use a predicate that answers the question of whether a route exists independently from the algorithm under test. Note that, in the absence of topology changes, the predicate of whether a path exists can be calculated beforehand. Also transmission errors have an influence on the existence-predicate, since a path temporarily does not exist in these cases.

For the dynamic case, we need to check the existence of a path between sender and receiver at any step in the algorithm. However, since checking this property requires an algorithm that visits all nodes (e.g., using the spanning-tree algorithm), this increases the state-space. In order to evaluate this predicate effectively (without increasing the state space), a suitable implementation of this predicate would be a *Maude* function. However, *Creol* does not currently allow interfacing to *Maude* functions. Therefore, the implementation ordinary *Creol* code is necessary.

Practically, the following check should be performed: for every route to be established, use existence-predicate; when a path exists the route must be found in a given number of steps; messages must be delivered within a certain amount of steps, e.g., in three times the path length.

Another property to evaluate is whether a route is re-established after a transmission error, given a path still exists. We also evaluate how long the path is interrupted after a transmission error occurs.

2.4.2 The loop-freeness property (b)

A routing loop is a situation where the entries in the routing tables form a circular path, thus preventing packets from reaching the destination. AODV uses sequence numbers to prevent routing loops[8], and is said to be loop-free.

The invariant for loop-freeness (Musuvathi et al., 2002) of AODV must be valid for all nodes. It uses *sequence numbers* of adjacent nodes, and the number of hops in the routing tables. In particular, for a route from s to d at a and b, b being the next hop to d from a, and using seq for the sequence number, and hcnt for the hop count, the equation $(\mathrm{seq}_a < \mathrm{seq}_b) \vee (\mathrm{seq}_a = \mathrm{seq}_b \wedge \mathrm{hcnt}_a > \mathrm{hcnt}_b)$ must be valid.

We implemented this by checking the above property every time a message is transmitted between nodes. To do this the network-object calls a routine that checks the MPCED property in an assertion. However, this assertion is complex, contains nested loops, and is expressed as code in the model. For model checking, the implementation of this assertion as a function in *Maude* would be preferable.

2.4.3 Single-sensor challenge-response properties (c)

The reaction of one node under test is checked using this property. Messages are sent to the node under test, and the responses from this node are matched against all correct responses. The correct responses are extracted from specifications or traces from running a system from simulations using different implementations.

The responses from the node under test might be different from the traces with respect to

- Sequence of messages; for example, when one challenge causes several responses. Note that this also applies for multiple challenges, which can cause a variety of responses.
- Each response can carry a variety of correct responses with respect to their content. The different fields of the message content can have values that do not necessarily match one single value, rather than following certain predicates. For example, the property for the sequence number does not imply a unique value.

There are several single-sensor properties that can be checked; these detail-properties express certain behaviour after a challenge; or the absence of a certain behaviour after a challenge. These properties include (list derived and adapted from the VEREOFY-case):

(i). **Always send with own ID:** All messages sent contain the sender id in the `sndNode` field.

8. Wiebling et al. (Wibling et al., 2004) show an example where AODV can produce a routing loop; however this algorithm is not identical to the AODV algorithm of RFC3561 (Perkins et al., 2003).

(ii). **Message leads to valid route:** Receiving a message from a neighbour node triggers an update in the routing table and ends in a valid route to the neighbour.

(iii). **RREQ with no valid route leads to RREQ broadcast:** Receiving an RREQ for a node for which no route is known triggers re-broadcasting the RREQ message to all neighbours.

(iv). **RREQ for this node leads to targeted RREP:** Receiving a RREQ for the node itself will be answered by a RREP message.

(v). **RREP leads to route to originator:** When receiving a RREP message from another node with route information to the sink, the update of the routing table leads to a valid route to the sink.

(vi). **RREP is re-broadcast:** RREP messages are being rebroadcast eventually. Holds only if its not from the sensor node itself and max hops not reached.

(vii). **Sending if route known only:** Unicasting a message takes place if and only if there is a valid route for the target ip. Holds only for non RERR messages, because there is no neighbour update when receiving miss-addressed data message.

(viii). **Routing table integrity:** Whenever there is a valid route for some node, there is also a valid route for the node which is designated as next hop target.

(ix). **All messages for the sink:** All data messages are addressed to the sink. Holds only for own packages, since received data packages might be faulty.

(x). **Processing does not require receive:** Processing does not require receiving messages.

(xi). **Increasing sequence number:** The own sequence number never decreases.

(xii). **Neighbour update triggers:** Each RREQ and RREP messages triggers a neighbour update on the routing table.

(xiii). **Updates terminate:** All updates terminate.

(xiv). **Update success:** Updating the routing table after receiving an AODV message leads to a valid route.

(xv). **Only one RREQ each:** There is only one RREQ message per data message.

(xvi). **Receive only in IDLE mode:** Receiving messages is only allowed when the sensor node is in its IDLE state.

2.4.4 The shortest-path property (d)

Here, we investigate whether the AODV algorithm finds the shortest path for the paths between source and sink node; also other metrics for paths could be checked. AODV finds the shortest path in the case of no packet loss. However, in the case of packet loss we can imagine cases where paths are found that do not fulfil the shortest-path property.

To check this property we count the number of hops that each payload-message takes from the source to the sink, and compare this number with the shortest existing path between source and sink.

2.4.5 The no-deadlock property (e)

Assuming an environment that can send all possible messages, the sensor node will never deadlock (robustness).

(xvii). **Deadlock in node:** Deadlock appears in the node, so that further requests cannot be handled.

(xviii). **Deadlock in protocol:** Deadlock appears in the protocol.

(xix). **Deadlock in model:** Deadlock appears in the model (modelling error; e.g., wrong use of synchronous or asynchronous calls in *Creol*).

2.4.6 Miscellaneous composed-system properties (f)

For the composed system, the following properties can be checked. Note that this list is neither complete, nor do all of these properties apply for all arrangements.

(xx). **Once a route is valid, it stays valid:** Routes do not get invalid in the static model without communication errors.

(xxi). **In the end, only data messages are transmitted:** In the static model without communication errors, from some moment onwards no AODV messages will be transmitted, but data messages only.

(xxii). **No RERR:** In the static model without communication errors there is no need for RERR messages to be handled.

(xxiii). **No useless RREQ messages:** When receiving an RREQ message for a node where a route is known, no additional RREQ messages are being created or forwarded.

(xxiv). **RREQ leads to RREP:** Sending a RREQ will lead to either receiving a reply, or an error message.

(xxv). **# of messages received:** Number of messages received at sink.

(xxvi). **packet loss:** Packet loss rate for data packets. If the packet loss rate is 0, then all data packages sent arrive at the sink.

(xxvii). **timing properties:** e.g., evaluating the number of periods until all messages are received.

(xxviii). **network connectivity:** check whether given nodes are connected.

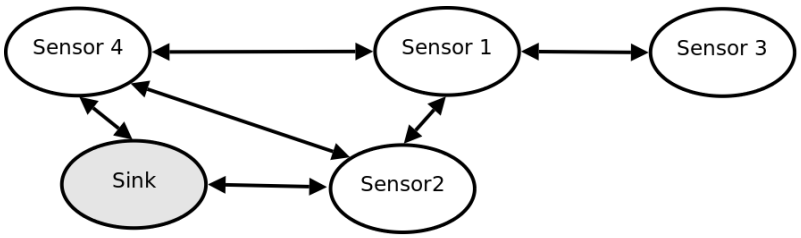


Figure 2. The network used as example in our simulations.

2.5 Non-Functional Properties (xxiv)

Several non-functional properties from the application domain can be evaluated, such as timing, throughput, delivery ratio, network connectivity, energy consumption, memory and buffer sizes, properties of the wireless channel, interferences, mobility (Salden et al., 2008), or other QoS properties. Each of these properties can be validated under different conditions, e.g., with/without packet loss, energy consumption, etc.

3 Simulation of a network

For our evaluation of the properties we used simulation using mainly techniques such as auxiliary variables, and assertions. Most of our experiments used a network with symmetrical communication via four sensor nodes and one sink node, as shown in Figure 2. We simulated the AODV model using various arrangements in order to validate the model for different situations. We looked at reliable networks, lossy networks, timeouts, energy consumption, and timed modelling. We also checked selected properties from classes (a), (b), (d), (e), and (f) that are suited for simulating the composed network. We present some of the evaluations below.

Reliable communication: As long as the network is connected[9], the evaluations showed that the modelled AODV algorithm fulfils the properties (a), (b), (d), (e), and (f). We emphasised on the evaluation of packet loss (f).(xxvi), and loop-freeness assertion (b). Other predicates for loop-freeness were also used (which failed as expected), and small, faulty changes in the model were introduced (which led to expected failures of Property (b)). The shortest path property (d) was fulfilled in all simulated occasions.

Lossy communication: When simulating lossy communication, both for singlecast, and for broadcast messages the packet loss rate (f).(xxvi) increases as expected. We also observed an increased number of RREQ and RREP messages in the system, using auxiliary variables. The shortest path property (d) was fulfilled in all simulated occasions.

In one occasion we could observe that the loop-freeness property (b) was not fulfilled using lossy communication. However, we have not investigated the reason of

9. We also simulated networks that are not connected, which behaved as expected.

this failure. We have saved the configuration for further investigation, and the state stored in the *Maude* file.

Re-sending lost messages with timeouts: The model allows re-sending of lost RREQ messages up to a certain number of times, using a timeout mechanism. We could observe that this mechanism decreased the packet loss rate (f).(xxvi), but at the same time this mechanism does not avoid all packet loss.

Energy consumption: Using the energy consumption arrangement we can force a communication failure of certain nodes after some actions. Using this arrangement we can study the re-routing behaviour in detail. We also studied the packet loss rate (f).(xxvi) for this arrangement.

Timed model: Using the timed model we can study the number of time steps needed for sending messages, as well as controlling the number of actions being performed simultaneously. We observed that the packet loss rate (f).(xxvi) is different to the untimed case, which is expected.

Using the timed model, we could observe a model deadlock (e).(xix), which is caused by the way the model is implemented, and certain properties of the current implementation of the *Creol* runtime system. This observation made changes in the model implementation necessary using asynchronous method calls.

We did not evaluate the properties (f).(xxi), (f).(xxiii), and (f).(xxiv), since it is necessary to store all messages during the simulation. However, such an arrangement will lead to a high number of states (state explosion).

4 Component Testing of One Node

For component testing we use one node under test with the same code as for holistic testing. However, we replace the network and all the other nodes using a test harness, as shown in Figure 3. The test is then performed by studying the output messages of a node when given input messages are applied.

Communication between nodes happens always through the interfaces of the network object which in term communicates with other nodes. Specifically a node does not communicate directly with another node. This is used to construct the test harness, which consists of a modified network object. The test harness has, and calls the same interfaces as the network object.

4.1 Test harness

The task of the test harness is to send messages to the interfaces of the node under test, and to observe its answers. Input messages to the node under test are extracted from traces extracted from running real systems or other simulations. The received answers are matched against the same traces.

Property	Description	Evaluation	S	T
(a)	Correct Operation	yes; for some arrangments.	•	
(b)	Loop-Freeness	yes	•	
(c)	Sgl-sensor challenge-resp.	yes		•
(c).(i)	always send with own ID	yes, as invariant during other tests		•
(c).(ii)	msg leads to valid route	yes (inferred from other tests)		•
(c).(iii)	RREQ w/o route⇒RREQ bc.	yes		•
(c).(iv)	RREQ for me leads to RREP	yes		•
(c).(v)	RREP triggers route to originator	yes		•
(c).(vi)	RREP is rebroadcasted	yes		•
(c).(vii)	send if route known	yes (no send if route unknown)		•
(c).(viii)	routing table integrity	no		○
(c).(ix)	all msg for sink	yes		•
(c).(x)	processing without receive	yes (during other tests)		•
(c).(xi)	increasing sequence number	yes, as invariant during other tests		•
(c).(xii)	neighbour update triggers	n/a (not accessible in black-box test)		
(c).(xiii)	updates terminate	yes (implicitly during other tests)		•
(c).(xiv)	update success	yes (implicitly during other tests)		•
(c).(xv)	Only one RREQ	n/a (needs timed interpreter)		
(c).(xvi)	Rec. in IDLE mode	n/a		
(d)	Shortest-Path	yes	•	
(e)	Deadlock-Freeness	partially	⊙	
(e).(xvii)	node deadlock	no		
(e).(xviii)	protocol deadlock	yes	•	
(e).(xix)	model deadlock	yes	•	
(f)	Misc. Composed-System	yes	•	
(f).(xx)	route stays valid	yes	•	
(f).(xxi)	only data msg	possible, not done	○	
(f).(xxii)	NoRERR	yes	•	
(f).(xxiii)	no useless RREQ	possible, not done	○	
(f).(xxiv)	RREQ triggers RREP	possible, not done	○	
(f).(xxv)	# msg.rec.	yes	•	
(f).(xxvi)	packet loss	yes	•	
(f).(xxvii)	timing properties	partially	⊙	
(f).(xxviii)	network connectivity	yes	•	
(f).(xxiv)	parameter tuning	partially	⊙	

Table 1. Properties evaluated in *Creol*; note that *S* marks Simulation, while *T* marks Testing as the method of choice.

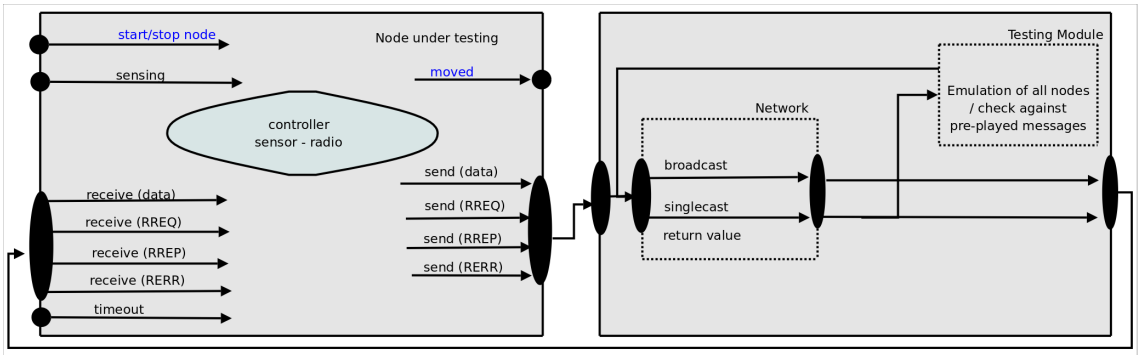


Figure 3. Testing of one node using the network object as a harness.

Although incoming broadcast, singlecast and outgoing packets (both from the perspective of the test harness) involve invoking different methods, the *Creol* language with its object-level parallelism makes it easy to encode a test case as a single sequential list of statements (as opposed to a less readable state machine with control flow distributed between different method bodies). Incoming messages are stored in a one-element buffer; the test case simply performs a blocking read on that buffer when waiting for a message from the object under test, before sending out the next message to the object. In that way, both creating a test case by hand and generating test cases from recorded traces become feasible.

A test verdict is reached by running the test harness in parallel with the object under test. If the test harness *deadlocks*, it expects a message from the object under test that is not arriving – in that case, a test verdict of *Fail* is reached. The other reason for test failure is an incoming message that does not conform to the expectations of the test harness; e.g. by being of the wrong type or having the wrong content.

A test verdict of *Success* is reached if the test harness completes the test case and the object under test conforms to the tester's expectations in all cases.

4.2 Traces

In addition to domain-specific single-object properties that have to be tested, test cases can be generated from the *Vereofy* model as well. In this subsection we outline the test generation process.

The traces from *Vereofy* are as follows: One step in the *Vereofy* trace consists of the content of all variables (within the nodes and buffers in the network), before and after each step, and the exchanged data. The messages in the trace are defined as follows (using a struct):

```
TYPE message_t = struct {
  // determine the type of the message
  message_type_t      message_type;
  id_t                dest_id;
  // encapsulation of sender and receiver IDs
  address_t           to_ip;
  id_t                from_ip;
```

```
  // case 1: sending aodv messages
  hop_counter_t      hop_count;
  seq_no_t           dest_seq_no;
  id_t               orig_id;
  seq_no_t           orig_seq_no;
  Bool               unknown_seq_no;
  // no TTL and XFlag for AODV.
  // case 2: for sending data messages
  data_type_t        the_data;
};
```

When the state information is removed we receive a sequence of messages that are exchanged simultaneously. In the following example Node 1 sends three RREQ messages to find a route to the Sink (Node 0). The RREP generated by the Sink does not arrive, since it remains in a buffer. This error trace we found by using *Vereofy*, and a newer version of the model no longer contains the error (the RREQ is sent several times).

```
{send[1]={RREQ,0,2,1,0,0,1,1,1,data0}}
{receive[0]={RREQ,0,2,1,0,0,1,1,1,data0}}
{send[0]={RREP,1,1,0,0,1,0,0,0,data0},send[1]={RREQ,0,2,1,0,0,1,2,1,data0}}
{receive[0]={RREQ,0,2,1,0,0,1,2,1,data0}}
{send[1]={RREQ,0,2,1,0,0,1,3,1,data0}}
```

In the tester for Node 1, written in *Creol*, the run-method waits for the messages denoted as `send[1]`, and sends messages denoted as `receive[1]`. Note that the trace can contain messages that are sent simultaneously, such as in Line 3 of the above trace. Note that for synchronous communication the calls of `send[0]={RREP,1,1,...}` and `receicve[1]={RREP,1,1,...}` take place simultaneously.

Traces received from the node under test are tested against *message patterns*, i.e. we abstract away details that could lead to spurious test failures not expressing a malfunctioning system. For example, the message sequence number can be chosen by the node, the only requirement is that it be monotonically increasing. This property is checked using an invariant in the tester, but a different concrete message number than that used by the *Vereofy* model cannot lead to test failure.

5 Conclusion

We presented the evaluation of a *Creol* model of AODV as a basis for the evaluation process of this model. We introduced the dimensions of *techniques*, *perspectives*, *arrangements*, and *properties* for this evaluation. The functional properties used for this evaluation were divided into five property classes (a) to (f). All these properties were aligned with the properties used to evaluate the *Vereofy* tool (Baier et al., 2009) for a later comparison. We performed network simulations of the composed system, and component testing of a single node in order to evaluate these functional properties.

Using the network simulation, several arrangements were evaluated, where most of

the properties were fulfilled as expected. On some occasions, we found properties that did not validate in the simulation, either due to bugs in the model, properties of the modelled AODV algorithm, artificially introduced bugs in the model, or property variants that are not supposed to validate successfully. In one occasion, we could detect deadlocks in the model in a timed-model arrangement; which could be recognised and fixed after on.

Using component testing, we validated the correct behaviour of a single node against properties extracted from the specification of the AODV algorithm. No deviations from specified component behaviour were identified in this process, which is unsurprising since components had already been extensively used for simulation and animation during initial model development at that point in time. However, the test suite served as an excellent help in regression testing during subsequent changes and extensions of the model.

We modelled a highly distributed application with many autonomously acting objects (sensor nodes). Evaluating the properties of the AODV algorithm, we encountered several challenges, such as modelling the suitable abstraction, using the suitable language constructs of *Creol*, and observing the properties from a suitable perspective. The major challenge when evaluating the AODV algorithm form a network perspective is to avoid a high number of states (state explosion) in the underlying interpreter.

We see that the properties suitable for component testing are rather disjunct from the properties suitable for network simulation. Therefore, these techniques are complementary to each other.

We found the *Creol* language and the tools useful in the evaluation of the AODV algorithm, and to gain insight in how complex algorithms like AODV work. We observed how small changes in the algorithm, and in the chosen arrangement, imply changes in its behaviour. We also detected the breach of certain properties, which will lead to further investigation of the reasons, removal of this misbehaviour, and, eventually, to a better understanding of AODV, and the algorithms used for sensor networks,

Acknowledgements

This research is part of the EU project IST-33826 CREDO: *Modeling and analysis of evolutionary structures for distributed services*. The authors want to express their thanks to their colleagues involved in the *Credo* project for their support during this work, especially Sascha Klüppelholz, Joachim Klein, Immo Grabe, Bjarte M. Østvold, Xuedong Liang, Marcel Kyas, Martin Steffen, and Trenton Schulz.

References

Akkaya, K. and Younis, M. (2005). A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3:325–349. 5

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422. 5

Al-Karaki, J. N. and Kamal, A. E. (2004). Routing techniques in wireless sensor networks: a survey. *IEEE Wireless Communications*, 11:6–28. 5

Baier, C., Blechmann, T., Klein, J., and Klüppelholz, S. (2009). Formal Verification of Components and Connectors. In *Proceedings FMCO'08*, volume 5751 of *Lecture Notes in Computer Science*. Springer-Verlag. 20

Chakeres, I. and Belding-Royer, E. (2004). AODV routing protocol implementation design. In *Proc. the 24th International Conference on Distributed Computing Systems Workshops(ICDCS'04)*, pages 698– 703, Tokyo, Japan. 5

Chiyangwa, S. and Kwiatkowska, M. Z. (2005). A timing analysis of AODV. In Steffen, M. and Zavattaro, G., editors, *FMOODS*, volume 3535 of *Lecture Notes in Computer Science*, pages 306–321. Springer. 7

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Quesada, J. F. (2001). Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*. 6

Gomez, C., Salvatella, F., Alonso, O., and Paradells, J. (2006). Adapting AODV for IEEE 802.15.4 mesh sensor networks: Theoretical discussion and performance evaluation in a real environment. In *Proc. IEEE the 7th International Symposium on World of Wireless, Mobile and Multimedia Networks (WOWMOM'06)*, pages 159–170, Niagara-Falls, Buffalo-NY, USA. 5

Grabe, I., Jaghoori, M. M., Aichernig, B., Blechmann, T., de Boer, F., Griesmayer, A., Johnsen, E. B., Klein, J., Klüppelholz, S., Kyas, M., Leister, W., Schlatte, R., Stam, A., Steffen, M., Tschirner, S., Liang, X., and Yi, W. (2009). Credo methodology. Modeling and analyzing a peer-to-peer system in Credo. In *3rd International Workshop on Harnessing Theories for Tool Support in Software*. 6, 8

Johnsen, E. B. and Owe, O. (2007). An asynchronous communication model for distributed concurrent objects. *Software and Systems Modeling*, 6(1):35–58. 6

Johnsen, E. B., Owe, O., Bjørk, J., and Kyas, M. (2007). An object-oriented component model for heterogeneous nets. In de Boer, F. S., Bonsangue, M. M., Graf, S., and de Roever, W. P., editors, *FMCO*, volume 5382 of *Lecture Notes in Computer Science*, pages 257–279. Springer. 7

Kyas, M. (2009). *Creol Tools User Guide*. Institutt for Informatikk, Universitetet i Oslo, Postboks 1080 Blindern, 0316 Oslo, Norway, 0.0n edition. 6

Leister, W. (2009). CreolE — a pragmatic extension to Creol. Note DART/06/09, Norsk Regnesentral. 6

Leister, W., Liang, X., Klüppelholz, S., Klein, J., Owe, O., Kazemeyni, F., Bjørk, J., and Østvold, B. M. (2009). Modelling of biomedical sensor networks using the creol tools. Report 1022, Norsk Regnesentral, Oslo, Norway. available from publ.nr.no/5042. 5, 7

Mozumdar, M. M. R., Gregoretti, F., Lavagno, L., Vanzago, L., and Olivieri, S. (2008). A framework for modeling, simulation and automatic code generation of sensor network application. In *Proc. Fifth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON 2008, June 16-20, 2008, San Francisco, USA*, pages 515–522. IEEE. 6

Musuvathi, M., Park, D. Y. W., Chou, A., Engler, D. R., and Dill, D. L. (2002). CMC: a pragmatic approach to model checking real code. In *OSDI, Usenix*. 6, 7, 13

Ölveczky, P. C. and Thorvaldsen, S. (2007). Formal modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude. In Bonsangue, M. M. and Johnsen, E. B., editors, *FMOODS*, volume 4468 of *Lecture Notes in Computer Science*, pages 122–140. Springer. 7

Perkins, C., Belding-Royer, E., and Das, S. (2003). Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental). Available from: http://www.ietf.org/rfc/rfc3561.txt. 5, 13

Salden, A., Stam, A., Balasingham, I., Steffen, M., Kyas, M., Leister, W., Liang, X., and Østvold, B. M. (2008). Credo deliverable 6.1: User driven requirements — addendum. Addendum to Deliverable D6.1, EU IST project, number 33826. 16

Stojmenovic, I. (2008). Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions. *IEEE Communications Magazine*, 46(12):102–107. 6

Tschirner, S., Liang, X., and Yi, W. (2008). Model-based validation of QoS properties of biomedical sensor networks. In *EMSOFT '08: Proceedings of the 7th ACM international conference on Embedded software*, pages 69–78, New York, NY, USA. ACM. 7

Wibling, O., Parrow, J., and Pears, A. N. (2004). Automatized verification of ad hoc routing protocols. In de Frutos-Escrig, D. and Núñez, M., editors, *FORTE*, volume 3235 of *Lecture Notes in Computer Science*, pages 343–358. Springer. 7, 12, 13

Wibling, O., Parrow, J., and Pears, A. N. (2005). Ad hoc routing protocol verification through broadcast abstraction. In Wang, F., editor, *FORTE*, volume 3731 of *Lecture Notes in Computer Science*, pages 128–142. Springer. 7, 12

Yu, I. C., Johnsen, E. B., and Owe, O. (2006). Type-safe runtime class upgrades in Creol. In Gorrieri, R. and Wehrheim, H., editors, *Proc. 8th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'06)*, volume 4037 of *LNCS*, pages 202–217. Springer-Verlag. 6