Analysis of Wireless Sensor Network Protocols in Dynamic Scenarios *

Cinzia Bernardeschi[1], Paolo Masci[1,2], and Holger Pfeifer[3]

[1] Department of Information Engineering, University of Pisa, Italy
cinzia.bernardeschi@ing.unipi.it
[2] Institute of Information Science and Technologies, CNR, Pisa, Italy
masci@isti.cnr.it
[3] Institute of Artificial Intelligence, Ulm University, Germany
holger.pfeifer@uni-ulm.de

Abstract. We describe an approach to the analysis of protocols for wireless sensor networks in scenarios with mobile nodes and dynamic link quality. The approach is based on the theorem proving system PVS and can be used for formal specification, automated simulation and verification of the behaviour of the protocol. In order to demonstrate the applicability of the approach, we analyse the reverse path forwarding algorithm, which is the basic technique used for diffusion protocols for wireless sensor networks.

1 Introduction and Motivation

Wireless devices have a limited transmission range and multi-hop communication protocols must be adopted when the network has a physical extension which exceeds the transmission range of nodes. Wireless Sensor Networks (WSNs) represent an example of wireless networks that are gaining more and more attention from the research community. In particular, WSNs are distributed systems consisting of a large number of spatially distributed, autonomous and cooperating nodes. The nodes of the network, referred to as *sensor nodes*, are battery-operated devices which provide limited computation capabilities, low-rate and low-range wireless communication, and are equipped with a number of sensors and actuators to monitor physical or environmental conditions [1].

Protocols for wireless networks are difficult to test on real devices, and simulation is currently the main technique used to investigate protocol behaviour. Software-based simulators are widely used to provide controlled environments in which experiments are to yield reproducible results. Protocols are commonly analysed with ad hoc simulators built on top of readily available network simulators, such as *Omnet++* [2], or distributed system simulators, such as *ptolemy* [3]. Currently, there is no established standard simulation framework.

* This work was partially supported by the European Commission through the Network of Excellence ReSIST (IST-026764). *This copy has been generated by the authors; the original publication is available at www.springer.com*

Formal modelling is of outstanding importance for reasoning about the behaviour of systems, and formal analysis methods are widely accepted as a method to provide additional confidence in the correctness of a system. For wireless sensor networks, there is increasing interest in using formal methods to verify key properties of popular routing algorithms [4], to evaluate protocol performance [5], and to validate simulation results [6].

In order to analyse protocols for wireless sensor networks, in this work we build on a framework [7] based on the *Prototype Verification System (PVS)*. PVS is a formal tool that combines an expressive specification language with an interactive theorem prover and it has been successfully employed for formal reasoning in several application domains (see [8] for an overview). The framework [7] allows an easy specification of the characteristics of wireless networks, such as limited communication range and lossy transmissions. In this work we introduce in the framework mechanisms to automate the analysis of dynamic scenarios with mobile nodes and quality changes of communication links. With our approach, a high-level clear description of the protocol can be developed. The formal specification can be animated and conveniently used to debug the specification and to obtain quantitative evaluations. Moreover, the approach opens the possibility of formally proving the correctness of the specification with respect to properties of interest.

In order to demonstrate the applicability of the approach, we show its application to the reverse path forwarding (RPF) algorithm, which is the basic technique used for diffusion protocols for WSNs. We employ a mechanism provided by the framework to automatically translate the formal specification into executable code, and simulate the algorithm in dynamic scenarios with mobile nodes and degraded / faulty wireless links. Moreover, by using the theorem prover of PVS, we prove that our specification satisfies desired properties of interest when the routing table is static, i.e., when the routing table is guaranteed to remain unchanged during protocol execution.

2 Basic Concepts of the Formal Framework

The framework presented in [7] combines formal verification and simulation to build an integrated approach that can be employed to improve the confidence in the behaviour of a system. The framework relies on the *Prototype Verification System (PVS)* [8].

2.1 PVS

The Prototype Verification System (PVS) is a specification and verification system which combines an expressive specification language with a powerful automated theorem prover. The PVS specification language builds on classical typed higher-order logic with the usual base types, `bool`, `nat`, `integer`, `real`, among others, and the function type constructor `[A -> B]`. Predicates are simply functions with range type `bool`. The type system of PVS also includes record types,

dependent types, and abstract data types. The most powerful concept are *predicate subtypes*, which can be used to check for violations, such as division by zero, or to express complex consistency requirements.

PVS specifications are packaged as *theories* that can be parametric in types and constants. Theorems and lemmas contained in PVS theories can be formally proved using the theorem prover of PVS. A built-in prelude and loadable libraries provide standard specifications and proved facts for a large number of theories. A theory can use definitions and theorems of another theory by importing it.

PVS also provides a *ground evaluator* [9] that can be used to animate functional specifications i.e., to translate the formal specifications into executable code. Indeed, although the specification language of PVS is based on higher-order logic and features a rich type system, a large subset of it is executable. The ground evaluator translates the executable constructs of PVS into efficient *Lisp* code. Furthermore, in order to still be able to simulate theories that also involve declarative specifications, the ground evaluator can be augmented by so-called *semantic attachments*, through which the user can supply pieces of Lisp code and attach them to the declarative parts. Using this mechanism, the *PVSio* package [10] extends the ground evaluator with a predefined library of imperative programming language features such as side effects and input/output operations, and also provides a high-level interface for writing user-defined semantic attachments.

2.2 Modelling and Analysing Network Protocols

The formal specifications of a network protocol consists of a collection of PVS theories. A PVS theory may represent a service installed on a node (e.g., packet logger, clock), a structural property of the network (e.g., network graph), a communication functionality (e.g., packet forwarding). For each PVS theory, a number of different versions can be provided in order to specify and analyse algorithms under several perspectives and at desired level of detail. The most abstract theory provides the declaration of types for a minimum set of mandatory attributes and the declaration of functions. More detailed theories can be derived from the abstract definition by specifying the behaviour of functions and by extending types.

In the following, we recall the basic aspects of the framework. For a more complete description of the theories, we refer to [7].

Nodes and Network Structure. Nodes in the network are identified by a unique identifier `node_id`. The base station has a special identifier `base_station`. The network is represented with a directed graph, and the external library [11] is used to benefit from various concepts and proved facts. To simplify graph specification, an auxiliary topology function is defined, which identifies, for each node, the set of neighbouring nodes.Once the topology is given, the network graph can be instantiated with an utility function that transforms a topology into a directed graph.

Communication Primitives. Nodes can exchange packets. Communication primitives take into account the structure of the network and enable packet reception only for nodes in the communication range of the sender: if node x sends out a broadcast packet, it is received only by the neighbours of x. Ideal and lossy communication are modelled through special addresses. Basically, lossy addresses are functions that return a subset of nodes with respect to their ideal counterpart. A number of different single-hop primitives were modelled to ease the specification of communication protocols: *Inject*, which can be used to send out packets generated by nodes; *Forward*, which is suitable to relay packets previously received by nodes; *Drop*, which is used to discard received packets. Additionally, nodes are also allowed to perform the *Idle* transition, i.e., a transition in which nodes do not perform any operation on incoming or outgoing packets. The implemented primitives are suitable for unicast, multicast and broadcast communication.

Protocols. A protocol is specified as a cyclic procedure executed on a generic node. The specification of the protocol may use services installed on the node, and the protocol itself can be used to define new services. Examples of services are *packet logger*, which stores statistics about sent and received packets, *receive buffer*, which models the buffer that holds received packets waiting to be processed, and *node scheduler*, which abstracts the clock of nodes and the medium-access control mechanism with the sequence of nodes that execute the algorithm. The *state* of a node is defined by the services installed on the node. The *network state* maintains the state of all nodes in the network.

3 Modelling Dynamic Scenarios

In this section we present extensions of the framework which support the specification of dynamic scenarios. Specifically, we show mechanisms suitable to express mobility patterns of nodes, possibility of link quality changes, and to automate the generation of routing tables.

Node Mobility. Node mobility can be expressed with functions that change network connectivity with the following three steps: *i)* select a target direction among those allowed by topology, *ii)* determine the new set of neighbours of the mobile node, *iii)* return a new topology according to the actual parameters. The `node_mobility_th` theory shows the definition of such a function in PVS. Three auxiliary functions are used to implement the corresponding steps.

```
node_mobility_th: THEORY
 BEGIN
 IMPORTING network_graph_th

   %-- select a target direction
   select_target(s: finite_set[node_id]): node_id

   %-- generate the new set of neighbours for the mobile node
   new_neighbours(tp: topology, mobile_node, target_node: node_id): finite_set[node_id] =
       {n: node_id | (n /= mobile_node) AND (tp(target_node)(n) OR n = target_node)}
```

```
  %-- change topology tp according to the new neighbourhood of the mobile node
  change_topology(tp: topology)(mobile_node: node_id, nbs: finite_set[node_id]): topology =
      LET tp = remove_node(mobile_node, tp)
       IN add_node(mobile_node, nbs, tp)

  %-- node mobility function
  node_mobility(m: node_id, tp: topology): topology =
      LET target  = select_target(tp(m)), new_nbs = new_neighbours(tp, m, target)
       IN change_topology(tp)(m, new_nbs)

  %-- ... more definitions omitted
END node_mobility_th
```

The target direction of the mobile node can be selected through a set of rules. Rules depend on the mobility model, and they can be either deterministic or random. For instance, suppose that a node moves according to a random walk, i.e., the mobile node takes a decision about the direction option for the next step according to a random distribution. Random walk is a well-known searching technique for resource discovery in decentralised networks. In the framework, such a mobility pattern can be specified by means of a function `random_walk`, which moves the mobile node n times:

```
random_walk_th: THEORY
  BEGIN
  IMPORTING node_mobility_th

  random_walk(n: nat)(ng: network_graph): RECURSIVE network_graph =
    IF n = 0 THEN ng
    ELSE LET tp = node_mobility(mobile_node, new_topology(ng)), ng = new_network_graph(tp)
          IN random_walk(n-1)(ng) ENDIF
      MEASURE n
END random_walk_th
```

Theory `random_walk_th` can be used to study protocols in dynamic scenarios. In the following we show an example where a mobile base station moves according to a random walk pattern and periodically injects a new packet; sensor nodes execute the flooding protocol to diffuse packets.

```
mobile_scenario_th: THEORY
  BEGIN %--... imports and some declarations omitted

  mobile_scenario(n: nat)(net: network_state, ng: network_graph)
         (sched_grp: finite_set[node_id]): RECURSIVE network_state =
    IF n = 0 THEN net
    ELSE LET ng = random_walk(S)(mobile_node, ng), %-- move the base station of S steps
             scheduled_id = flooding_app_scheduler(sched_grp),
             net = IF scheduled_id = base_station
                   THEN inject(scheduled_id)(net, ng) %-- the base injects a new packet
                   ELSE flooding(scheduled_id)(net, ng) %-- nodes execute flooding
                   ENDIF
          IN mobile_scenario(n-1)(net, ng)(remove(scheduled_id, sched_grp))
    ENDIF
      MEASURE n

END mobile_scenario_th
```

Note that the specification above is executable and can be animated to perform simulation. Examples of random walks traced by the mobile node during simulation on a grid with 64 places with 8 columns and rows are shown in Figure 1. In our simulations, the mobile node is initially placed on the top-left corner

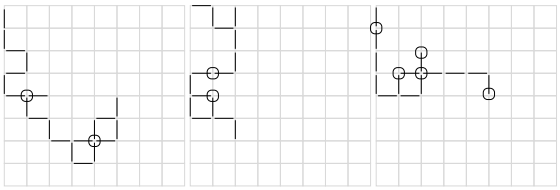


Fig. 1. Examples of random walks that can be generated during simulations. Circles represent places where the mobile node stopped for at least one step.

of the grid. Different initial positions can, of course, be chosen. Grids with larger number of places and different structure can be used as well. For the sake of simplicity, the figure reports only the trace drawn by the mobile node without any direction indication.

Automated Generation of Routing Tables. Routing tables are, by definition, tables that store, for each node, a path suitable to reach other nodes. Hence, we specified the routing table with a function which returns, for any node, a vector of paths: given a network graph G and a routing table rt, the vector of paths starting from i is $rt(i)$, and the path from i to j is $rt(i)(j)$. In this definition, we benefit from the definition of paths provided in [11]: a path from i to j is a `prewalk` of nodes (i.e., a sequence of nodes) that must be traversed on the network graph to reach j starting from i.

```
routing_table_th: THEORY
BEGIN
IMPORTING network_graph_th, digraphs[node_id]

   routing_table: TYPE = [i: node_id -> [j: node_id -> prewalk[node_id]]]
   routing_table?(rt: routing_table, g: network_graph): bool =
                                    FORALL (i, j: node_id): route?(g, rt(i)(j), i, j)

   valid_route?(g: network_graph, p: prewalk[node_id], i, j: node_id): bool =
       ((i /= j) AND (l(p) > 1) AND path_from?(g, p, i, j))

   valid_routing_table?(rt: routing_table, g: network_graph): bool =
       routing_table?(rt,g) AND FORALL (i, j: node_id): valid_route?(g, rt(i)(j), i, j)

   %-- ... more definitions omitted
 END routing_table_th
```

In order to automate the generation of routing tables, we extended the framework with a new theory `rtgen_th` which defines a service that, given a network graph, generates a routing table. Basically, `rtgen_th` models a technique that is frequently used in WSNs to generate routing tables. The technique is based on a protocol which performs the following actions. A node forwards a received packet only if the packet is received for the first time, otherwise the packet is dropped; packets carry a hop-count field in their payload that is incremented every time the packet is forwarded. To build a routing table with the above algorithm, the base station sends out a *beacon* packet with hop-count equal to 0; as the packet gets forwarded in the network, nodes learn about their own distance from the

base station, and nodes are also able to estimate the distance of their neighbours by inspecting the sender address and hop-count fields of the received packets.

In the framework, the service can be specified as follows:

```
rtgen_th: THEORY
 BEGIN %-- ...imports omitted

  rtgen(x:node_id)(net:network_state, g:network_graph): network_state =
    IF empty?(net_receive_buffer(net)(x)) THEN idle(x)(net, g)
    ELSE LET received_pk = getpacket(net_receive_buffer(net)(x)),
             hopcount = getpayload(received_pk)(COUNTER_FIELD)
         IN IF forwarded_packets(x, net_log(net)) = 0
            THEN forward(received_pk WITH [destination_addr := lossy_bcast_addr,
                                           payload := new_payload(hopcount+1)])
                        (x)(net,g)
            ELSE drop(received_pk)(x)(net, g)
            ENDIF
    ENDIF

  rtgen_rec(x: node_id)(net: network_state, g: network_graph): RECURSIVE network_state =
      IF empty?(net_receive_buffer(net)(x)) THEN idle(x)(net, g)
      ELSE LET net_prime = rtgen(x)(net, g) IN rtgen_rec(x)(net_prime, g)
      ENDIF
       MEASURE size(net_receive_buffer(net)(x))

  rtgen_service(net: network_state, g: network_graph):
         RECURSIVE network_state =
      LET grp = next_group(net_receive_buffer(net), net_log(net))
       IN IF zero?(grp) THEN net
          ELSE LET scheduled_id = random(grp),
                   net_prime = rtgen_rec(scheduled_id)(net, g)
               IN rtgen_service(net_prime, g)
        ENDIF
       MEASURE size(next_group(net_receive_buffer(net), net_log(net)))

   %-- ... more definitions omitted
 END rtgen_th
```

The routing table can be obtained from the information in the log of nodes at the end of the execution of `rtgen_service`. Routing tables can be generated either with ideal or lossy transmissions. In the case of ideal transmissions, routing tables always define a spanning tree with minimum hop distance from the base station. With lossy transmissions, on the other hand, the generated routing tables define a spanning tree with a random structure. For lossy transmissions, we can instantiate the probability P_{rx} for which node $x \in$ `lossy_bcast_addr`. When lossy transmissions are used, a `check` predicate controls if the generated routing table is valid. In the case that the routing table is not valid – this may happen with lossy transmissions when some nodes are not reached by the beacon packet sent by the base station to build the routing table – the procedure was instrumented to automatically try to generate a new routing table.

4 Case study: the Reverse Path Forwarding Algorithm

In this section we will apply the proposed approach to the reverse path forwarding (RPF) algorithm, which is the basic technique used for diffusion protocols for WSNs. RPF is a broadcast routing method which exploits the information contained in the routing table to deliver packets generated by a base station to

all other nodes in a multi-hop network. With RPF, packets are propagated with the following policy: a node n accepts a packet received from node p only if n believes that p is the best next hop on the path to the base station, as specified in the routing table. It is well known that, under the assumption of a static routing table, the reverse path forwarding algorithm delivers exactly one copy of the broadcast packet to all nodes. If, however, the routing table is dynamic, as is usually the case in real-world deployments, then such guarantees cannot be made for RPF [12].

In our framework, the algorithm is specified as follows: a broadcast packet received by node x is accepted for forwarding if the sender address of the packet is the best next hop of x towards the base station. The best next hop can be derived from the routing table. The specification of the algorithm is:

```
rpf_th: THEORY
  BEGIN IMPORTING routing_table_th    %-- ... more imports omitted

 rpf(x: node_id)(g:network_graph, base_station:node_id, rt:routing_table)
    (net: network_state): network_state =
   IF empty?(net_receive_buffer(net)(x)) THEN idle(x)(net, g)
   ELSE LET received_pk = getpacket(net_receive_buffer(net)(x))
            sender_addr = sender_addr(received_pk),
            next_hop = next_hop(x, base_station)(g, rt)
        IN IF sender_addr = next_hop
           THEN forward(received_pk)(x)(net, g)
           ELSE drop(received_pk)(x)(net, g)
           ENDIF
   ENDIF

   %-- ... more definitions omitted
 END rpf_th
```

The main property of the RPF algorithm is the following:

Property P: *If the routing table is correct and static, then exactly one copy of the broadcast packet sent by the base station will be delivered to all nodes in the network.*

5 Simulation

To simulate RPF, we need to specify an application scenario that takes into account how the system evolves. As RPF itself does not generate routing tables, we use the PVS function `rtgen` shown in Section 3 for that purpose.

In our simulations, we generated routing tables by using `lossy_bcast_addr` with $P_{rx} = 0.94$, which can be a reasonable value for low power wireless devices [13]. A semantic attachment of the PVSio library is used to generate pseudo-random numbers. Moreover, a theory for generating random sets of nodes has been implemented. In our simulations, a uniform distribution is used, but other distributions can be adopted as well, either providing an executable specification or changing the semantic attachment. On a desktop computer with a 2 GHz processor, a valid routing table can be generated in seconds for networks of 100 nodes placed on a grid. For a network of 1000 node placed on a grid, the average time is of few minutes.

In our setting, the base station periodically injects packets in the network, and other nodes apply the RPF algorithm. Each packet injected by the base station is uniquely identified: this way we can derive useful statistics by inspecting the log of nodes. In our simulations, a random node is scheduled at each simulator step. The scheduler is specified so that fairness of execution between nodes is guaranteed, i.e., all nodes are able to execute the algorithm and make progress at the same speed. Nodes operate in burst mode, i.e., when a node is scheduled, all packets in the receive buffer are processed according to the RPF algorithm.

We aim at evaluating the delivery ratio and the overhead due to duplicates in some representative scenarios of dynamic environment. The delivery ratio of a node x is the number of packets delivered to node x over the number of packets sent to node x. If the delivery ratio is one, then all packets were delivered to the intended destination. The overhead due to duplicates is the amount of traffic due to packet replicas. Such overhead can be caused by the RPF algorithm when the routing table is not static [12].

In the following paragraphs, we show simulation results obtained for networks of 64 nodes placed on a grid with 8 columns. Networks with larger number of nodes and different structures can be simulated as well.

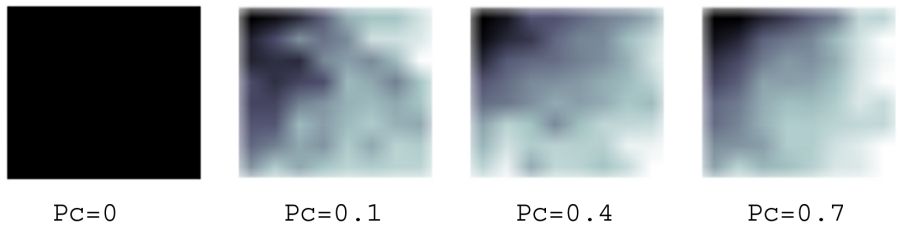
Link Quality

Link quality is a measure of the probability of successful communication over a link. In wireless networks, nodes can estimate the quality of the links with their neighbours, e.g., through the received signal strength intensity (RSSI), which is automatically computed by the wireless radio chip whenever a packet gets received [14]. In wireless environments, link quality may dramatically change because of many factors, ranging from hardware/software nodes failure to environmental factors (e.g., humidity, obstacles). In multi-hop wireless networks, link failures may lead to network partitioning because of the limited communication range of the radio equipment. Hence, whenever a link between two neighbouring nodes fails, such nodes must choose a new next hop in order to recover the routing table.

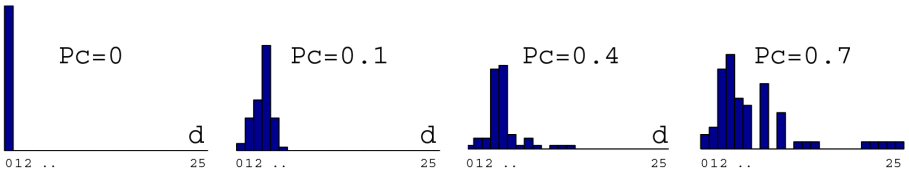
We evaluate the RPF algorithm in a scenario where the routing table always contains paths with the best local link quality. The base station periodically injects a new packet in the network, and all nodes apply the RPF algorithm to diffuse the packet. We assume that the quality of different links is not correlated, and that link quality may change with a uniform probability P_c. The specification of the application scenario used to simulate the network for n steps is shown in the following.

```
rpf_sim_th: THEORY
  BEGIN
   %-- ...imports omitted

link_quality_app(Pc:real)(n:nat)(net:network_state, ng:network_graph, rt:routing_table)
                (base_station:node_id, sched_grp:finite_set[node_id]):
        RECURSIVE network_state =
```



(a)



(b)

Fig. 2. Examples of simulation results with different link quality change probability Pc. (a) Delivery ratio at the end of different simulation runs. (b) Distribution of duplicates.

```
IF n = 0 THEN net
ELSE LET (ng, rt) = change_routing_table(Pc)(n)(base_station, ng, rt),
         sched_id = random_scheduler(sched_grp),
         net_prime = IF sched_id = base_station
                     THEN inject_service(sched_id,n)(net,ng)(rt)
                     ELSE rpf_service(sched_id)(net,ng)(base_station,rt)
                     ENDIF
      IN link_quality_app(Pc)(n - 1)(net_prime, ng, rt)
                          (base_station, update_sched(sched_id, sched_grp))
ENDIF MEASURE n
 %-- ... more definitions omitted
END rpf_sim_th
```

Function `change_routing_table` reflects the possible changes in the routing table due to changes in the quality of the links, and uses `rt_gen` to generate new routing tables.

The results of four simulations runs in which RPF has been evaluated with different probabilities Pc are shown in Figure 2. For each simulation run, routing tables were chosen randomly, and link quality changed if the value of a random variable was higher than a given threshold. Figure 2(a) shows a snapshot of the grid network at the end of simulation runs. The images relate the physical position of nodes with colours that highlight the number of packets successfully delivered (darker colours for higher delivery ratios). The base station is placed in the top-left corner of the grid. As expected, the delivery ratio is 1, i.e., all packets are delivered, when the link quality does not change (Pc=0). In the case of dynamic scenarios (Pc$\neq$0), on the other hand, some nodes are not able to receive the packet sent out by the base station. Moreover, we can notice that

the delivery ratio is relatively high for nodes that are closer to the source node (i.e., the base station), while it decreases rapidly for distant nodes. Figure 2(b) reports the distribution of duplicates among nodes. The x-axis of the histogram reports the number d of duplicates, the y-axis reports the number of nodes that received d duplicates. It can be noticed how the number of duplicates rapidly grows with the dynamics of the network.

These kinds of analyses can be applied to communication protocols to derive useful information about possibly unexpected behaviours. For instance, if energy consumption is a main concern for the application, results may point out that developers should combine their algorithm with a mechanism that efficiently suppresses duplicates.

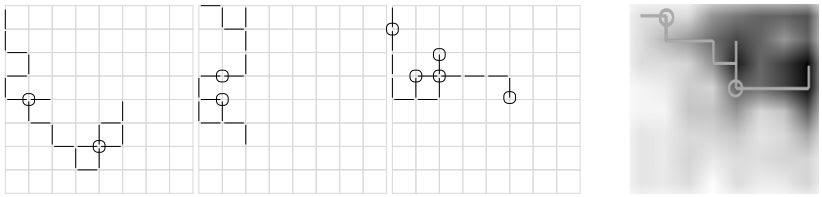
Node mobility

We also evaluated the RPF algorithm with changing routing table due to a mobile base station. In our scenario, the base station moves according to a random walk pattern, i.e., the base station takes a decision about the direction option for the next step according to a random distribution. Random walks are well-known searching techniques for resource discovery in decentralised networks. Every time the base station moves, a new routing table is generated and a new packet is sent out. The application used to simulate n steps of the network is similar to that of link quality changes. The main difference, with respect to the specification used for link quality changes, is the function used to generate the new routing table. In this case, first, the topology of the network is updated to reflect the mobility of the base station; second, a new routing table is generated:

```
rpf_sim_th: THEORY
  BEGIN %-- ...imports omitted

node_mobility_app(n:nat)(net:network_state, ng:network_graph, rt:routing_table)
                 (base_station: node_id, sched_grp: finite_set[node_id]):
         RECURSIVE network_state =
  IF n = 0 THEN net
  ELSE LET (ng, rt)  = move_base_station(n)(base_station, ng, rt),
           sched_id  = random_scheduler(sched_grp),
           net_prime = IF sched_id = base_station
                       THEN inject_service(sched_id,n)(net,ng)(rt)
                       ELSE rpf_service(sched_id)(net,ng)(base_station,rt)
                       ENDIF
       IN node_mobility_app(n - 1)(net_prime, ng, rt)
                            (base_station, update_sched(sched_id, sched_grp))
  ENDIF
MEASURE n
   %-- ... more definitions omitted
  END rpf_sim_th
```

Figure 3 shows some results of simulations where the base station was initially placed on the top-left corner of the grid. The random walks traced by the mobile base station during three simulation runs are shown in Figure 3(a). For the sake of simplicity, the figure reports only the trace drawn by the mobile node without any direction indication. Figure 3(b) shows the delivery ratio obtained for a simulation run (the walk performed by the mobile node is shown in overlay).



(a) (b)

Fig. 3. Example of simulations results with a mobile base station. (a) Random walks; circles represent places where the mobile node stopped. (b) Delivery Ratio; the random walk is shown in overlay.

It can be noticed that the delivery ratio is higher for nodes closer to the path traversed by the base station. The average delivery ratio is between 0.23 and 0.59, with an average value of 0.34; with more details, the average delivery ratio of nodes that were immediate neighbours of the mobile node is 0.4, while for other nodes it is only 0.29 in average. Such results are coherent with respect to the results presented in [15], where RPF was evaluated for a mobile base station following a random waypoint mobility pattern.

6 Formal Verification

In this section, we outline the proof that the specification of RPF satisfies property **P** when the routing table is static. The execution of the RPF algorithm is specified as a sequence of network states which starts from an initial state and repeatedly applies a state transition function. For RPF, the initial state models the injection of a packet in the network by the base station. The state transition function models the execution of the algorithm of a generic node x, which is applied recursively to all received packets using an auxiliary `rpf_service` function:

```
rpf_proof_th: THEORY
  BEGIN %-- ... imports omitted

   rpf_service(t: nat)(x: node_id)
              (ns:network_state, g:network_graph, rt:routing_table) : network_state =
     LET scheduled_node = scheduler(t),
         n = size(net_receive_buffer(ns)(scheduled_node))
      IN execute(rpf(scheduled_node)(g,base_station, rt))(n)(ns)

   rpf_transition(ns0, ns1: network_state, t: nat)
                 (g: network_graph, rt: routing_table): bool =
     ns1 = rpf_service(scheduled_node)(ns0, g)(base_station, rt)

   rpf_trace(seq: sequence[network_state])(g: network_graph, rt: routing_table): bool =
     seq(0) = initial_rpf_state(base_station) AND
      FORALL(t:nat): rpf_transition(seq(t),seq(t+1),t)(base_station,g,rt)

   %-- ... more definitions omitted
  END rpf_proof_th
```

The formal proof of property **P** is by an induction on the execution traces of RPF, i.e., on sequences that start with the initial state and apply the RPF transition function to generate subsequent states states. Furthermore, the proof makes use of the following properties and constraints, which can be expressed as additional lemmas, or sub-type conditions:

- the routing table is correct and does not change
- the network is not partitioned
- a correct routing table rt exists, which defines a spanning tree rooted at the base station
- all nodes are guaranteed to be scheduled at least once every N steps of the execution trace, where N is the number of nodes in the network
- all nodes operate in burst mode: when a node is allowed to transmit, it sends out all packets waiting to be transmitted

To accomplish the overall proof, the following lemmas proved useful:

Lemma 1. *For all execution traces and network states, for every node x, the number of packets with sender address equal to the next hop of x in the receive log of node x, is equal to the number of packets in the forward log of the next hop of x (by definition, the next hop of the base station is the base station itself).* ◇

Lemma 2. *For all execution traces and network states, for every node x, the number of packets with sender address equal to the next hop of x in the forward log of node x is less than or equal to the number of packets in the receive log of node x.* ◇

Lemma 3. *For all execution traces and all network states, for every node x, if the number of packets with sender address equal to the next hop of x in the receive log of x is ≥ 1 at time t, then the number of packets in the forward log of node x is ≥ 1 at time $(t+N)$, where N is the number of nodes in the network.* ◇

The delivery of the broadcast packet is assessed through the receive log of nodes. To this end, the proof of property **P** is split into two parts: first we prove that the number of received packets is at most one; second, we prove that the number of received packets is at least one. The proofs have been developed and mechanically checked with the theorem prover of PVS.

Theorem 1. *For every node x, the number of received packets with sender equal to next hop of x is at most one.*
Proof outline. *The proof is given by induction on the number k of hops between the node and the base station on the spanning tree defined by the routing table.*
Base*: $k = 1$. The proof follows from the assumption that the base station injects only one packet.*
Induction*: $k = n + 1$. The path p between the base station and node x is split into a path p' between the base station and next hop of x and an edge between next hop of x and x. The length of p' is n. Hence, the inductive hypothesis holds for the next hop of x, which receives at most one packet. By using Lemma 2, we obtain that x receives at most one packet from the next hop* ◇

Theorem 2. *For every node x, the number of packets with sender equal to next hop of x is at least one.*
Proof Outline. *The proof is given by induction on the number k of hops between the node and the base station on the spanning tree defined by the routing table and by using the assumption on fairness for transmissions.*
Base*: $k = 1$. By construction of the initial state, the base station has injected a packet at time $t = 0$. Hence the receive log of neighbours of the base station contains the packet sent by the base station at time $t = 0$.*
Induction*: $k = n + 1$. The path p between the base station and node x is split into a path p' between the base station and next hop of x and an edge between next hop of x and x. The length of p' is n. Hence, the inductive hypothesis holds for the next hop of x, which receives at least one packet at a time t. By using Lemma 3, we obtain that next hop of x forwarded at least one packet at time $t + N$. By Lemma 1, there is at least one packet with sender address equal to next hop of x in the receive log of x* ◇

7 Related Work and Conclusions

The need for formal modelling and analysis of algorithms for wireless networks has been pointed out in many papers. In [4], basic properties of the Reverse Path Forwarding algorithm have been analysed with FDR and Alloy Analyser. Scalability is the main problem of such an approach: only very simple and small network configurations were analysed, and specific hypotheses were assumed in a hand-proof of the correctness the algorithm . In [5], Lamport's Temporal Logic of Actions is used to model and simulate diffusion protocols for discovering routing trees for gathering and disseminating data. The analysis focuses on performance variation of push and pull phases of the diffusion protocol for routing trees with different shapes, however without the objective of algorithm design evaluation. In [16], Real-Time Maude has been applied to the OGDC density control algorithm and networks of several hundred nodes were analysed. The approach allows modelling the algorithm at high levels of detail, using broadcast and unicast communication primitives; results are claimed to be often more accurate compared to other network simulators.

In this paper we show an approach based on the PVS system to analyse protocols for WSNs in dynamic scenarios with mobile nodes and link quality changes. As case study, we developed a formal specification for RPF. Through simulation, we evaluated the algorithm and we have obtained results that are coherent with those reported in other papers. Furthermore, we used the theorem prover of PVS to verify core correctness properties of RPF when the routing table is guaranteed to remain unchanged.

The approach allows an easy specification of the characteristics of wireless networks, such as limited communication range, lossy transmissions, node mobility. The advantages of our approach are that it allows to develop a formal specification of the protocol at different levels of abstraction, opening the possibility to make complex systems tractable, and that the same formal specification

can be automatically translated into executable code suitable for simulations and as basis for formal reasoning in a theorem proving system.

References

1. Akyldiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: a Survey. Computer Networks **38** (2002) 393–422
2. Varga, A.: The Omnet++ Discrete Event Simulation System. Proceedings of the European Simulation Multiconference (ESM'2001) (June 2001)
3. Buck, J., Ha, S., Lee, E.A., Messerschmitt, D.G.: Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems. In: Readings in hardware/software co-design. Kluwer Academic Publishers (2002) 527–543
4. Bolton, C., Lowe, G.: Analyses of the Reverse Path Forwarding Routing Algorithm. In: Proc. Intl. Conf. on Dependable Systems and Networks, IEEE Computer Society (2004) 485–494
5. Nair, S., Cardell-Oliver, R.: Formal Specification and Analysis of Performance Variation in Sensor Network Diffusion Protocols. In: Proc. Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems, ACM (2004) 170–173
6. Bhargavan, K., Gunter, C., Lee, I., Sokolsky, O., Kim, M., Obradovic, D., Viswanathan, M.: Verisim: Formal Analysis of Network Simulations. IEEE Trans. Software Engineering **28**(2) (2002) 129–145
7. Bernardeschi, C., Masci, P., Pfeifer, H.: Early Prototyping of Wireless Sensor Network Algorithms in PVS. In: SAFECOMP '08: Proceedings of the 27th international conference on Computer Safety, Reliability, and Security, Berlin, Heidelberg, Springer-Verlag (2008) 346–359
8. Owre, S., Rushby, J., Shankar, N., v. Henke, F.: Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. IEEE Trans. on Software Engineering **21**(2) (1995) 107–125
9. Crow, J., Owre, S., Rushby, J., Shankar, N., Stringer-Calvert, D.: Evaluating, testing, and animating PVS specifications. Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA (2001)
10. Muñoz, C.: Rapid prototyping in PVS. Technical Report NIA Report No. 2003-03, NASA/CR-2003-212418, National Institute of Aerospace, Hampton, VA (2003)
11. Butler, R., Sjogren, J.: A pvs graph theory library. Nasa Technical Memorandum 1998-206923, NASA Langley Research Center, Hampton, Virginia (1998)
12. Dalal, Y., Metcalfe, R.: Reverse Path Forwarding of Broadcast Packets. Communications of ACM **21**(12) (December 1978) 1040–1048
13. Woo, A., Tong, T., Culler, D.: Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In: SenSys03, New York, NY, USA, ACM Press (2003) 14–27
14. Texas Instruments: Chipcon CC2420 Datasheet (2007) Available at `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`.
15. Clausen, T., Larsen, N., Olesen, T., Viennot, L.: Investigating Data Broadcast Performance in Mobile ad hoc Networks. In: The 5th International Symposium on Wireless Personal Multimedia Communications (WPMC). (2002)
16. Ölveczky, P., Thorvaldsen, S.: Formal Modeling and Analysis of the OGDC Wireless Sensor Network Algorithm in Real-Time Maude. In: Proc. Intl. Conf. on Formal Methods for Open Object-Based Distributed Systems. Volume 4468 of LNCS., Springer (2007) 122–140