Modelling and Analysing Routing Protocols Diagrammatically with Bigraphs

MARAM ALBALWE, University of Glasgow, UK and University of Tabuk, Saudi Arabia
BLAIR ARCHIBALD, University of Glasgow, UK
MICHELE SEVEGNANI, University of Glasgow, UK

As more end-user applications depend on Internet of Things (IoT) technology, it is essential the networking protocols underpinning these applications are reliable. Using Formal Methods to reason about protocol specifications is an established technique, but, due to their perceived difficulty and mathematical nature, receive limited use in practice. We propose an approach based on Milner's bigraphs—a flexible **diagrammatic** modelling language—that allows developers to "draw" the protocol updates as a way to increase use of formal methods in protocol design. To show bigraphs in action, we model part of the Routing Protocol for low-power and Lossy Networks (RPL), popular in wireless sensor networks, and verify it using model checking. We compare our approach with the more common simulation approach, and show that analysing the bigraph model often finds more valid routes than simulation (that usually returns only a single routing tree even with 500 simulations), and that it has comparable performance. The model is open to extension, with less implementation effort than simulation, and we show this through two examples: a security attack and physical link drops. Bigraphs seem a promising approach to protocol design, and this is the first step in promoting their use.

CCS Concepts: • **Networks → Protocol correctness**; • **Computing methodologies → Modelling and simulation**; • **Theory of computation → Models of computation**; • **Software and its engineering → Model checking**.

Additional Key Words and Phrases: Bigraphs, wireless sensor networks, routing, formal methods, model checking

ACM Reference Format:
Maram Albalwe, Blair Archibald, and Michele Sevegnani. 2024. Modelling and Analysing Routing Protocols Diagrammatically with Bigraphs. 1, 1 (July 2024), 27 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Internet of Things relies on interconnected devices, *e.g.* sensors and actuators, which is enabled through an array of networking protocols, often designed for specific device types, *e.g.* low-power devices. With this dependence on networking, it is essential the networking protocols underpinning these applications are reliable.

Formal methods are one approach to provide guarantees about reliability. While formal methods, and model checking in particular, have been widely used for rigorous system analysis both in industry and academia—thanks to a large number of tools [23, 16, 31, 19], variety of modelling languages [12], support for exhaustive analysis, and the existence of specialised tools for security analysis [30, 41]—they are often seen as difficult to use and uptake is slow. The trade-offs between the benefits of formal methods in guaranteeing behaviour and their usability is a pressing question in protocol

Authors' addresses: Maram Albalwe, m.albalwe.1@research.gla.ac.uk, University of Glasgow, 18 Lilybank Gardens, Glasgow, Scotland, UK, G12 8RZ and University of Tabuk, P.O.Box 741, Tabuk, Tabuk, Saudi Arabia, 71491; Blair Archibald, University of Glasgow, 18 Lilybank Gardens, Glasgow, UK, G12 8RZ, blair.archibald@glasgow.ac.uk; Michele Sevegnani, University of Glasgow, 18 Lilybank Gardens, Glasgow, UK, G12 8RZ, michele.sevegnani@glasgow.ac.uk.

design, with the Internet Engineering Task Force (IETF) recently proposing their *Usable Formal Methods Research Group*[1].

We believe diagrammatic modelling techniques, *i.e.* Milner's Bigraphs, can be practically applied to protocol design to usability without sacrificing rigour.

We show that Bigraphs can be employed to model and verify Wireless Sensor Networks (WSN) routing protocols, and also show how user-defined rules make it easy to experiment with the protocol, *e.g.* adding in new rules to model security aspects or wireless links instability. We showcase our approach by modelling and analysing RPL [3]—Routing Protocol for Low-Power and Lossy Networks as an example while we expect similar approaches to apply to a much wider range of network protocols as can be seen by similar models for the 802.11 MAC protocol [14], and other WSN topology protocols [7].

Bigraphs have been successfully used in modelling and verifying different systems such as ubiquitous systems [10], cyber-security in smart buildings [50], and biological processes [29]. A key feature of bigraphs is their diagrammatic notation that opens up formal modelling to a wider audience, including protocol designers who may be unfamiliar with the complex notations often found in formal modelling tools. Bigraphs also benefit from user-defined rewriting rules (as opposed to fixed rules in models like the π-calculus [36]) and the ability to quickly experiment with different initial states, *e.g.* by drawing the sensor network topology. As a graph-rewriting formalism based on relating entities both spatially, through nesting, and with non-local connectivity, bigraphs can mirror sensor networks that also have a strong spatial component, *e.g.* the radio range determining the physical neighbour set, and virtual connectivity, *e.g.* determining multi-hop paths from a sensor to a base station [7].

Wireless Sensor Networks (WSN) [28] use multiple sensors to gather a large amount of data over multi-hop points than a single sensor can. Low-cost sensors often have low-powered transmitters that are out of range of an internet gateway node meaning multi-hop routing—where data is sent from sensors to other sensors close to the gateway—is a necessity for most WSN applications. A key goal of WSN routing is to build an efficient route of communication between the gateway and each sensor to allow reliable data transfer. Given the resource-constrained nature of the sensors (usually small embedded systems), and the possibility of deploying them at harsh environments, *e.g.* sensors within a volcanic environment where radio transmissions are unreliable [54], sharing information effectively requires routing protocols specialised for low-power and lossy networks.

It essential to prove the protocols are correct, secure, and robust. Current engineering practice mainly focuses on evaluating protocols in real embedded devices through experiments within simulation/emulation environments such as Cooja [39] or directly on physical testbeds [27]. An advantage of these approaches is that the real *implementation* of the protocol, which may deviate from the protocol specification, is tested. However, these approaches are non-exhaustive and cannot make *guarantees* about protocol behaviour. They are also inflexible when trying to discover the effects of protocol changes as this requires potentially complex re-implementation work. Formal, mathematically-based, models of protocols can overcome some of these shortcomings by providing exhaustive analysis and proof.

We make the following research contributions:

- We show how bigraphs can model RPL's route construction, and provide an executable specification of the model in the BigraphER [44] framework.

[1] https://wiki.ietf.org/en/group/ufm

- We experimentally compare our bigraph model against the RPL-Lite[2] implementation in the Contiki-NG operating system [38]. Simulation is performed using the Cooja network simulator. We compare based on the running time and number of routes found.
- Using model checking, we show the RPL specification maintains useful properties, *e.g.* routes are valid.
- We discuss how the bigraph approach allows easy extension (*i.e.* without full implementation effort) by showing how to encode malicious attacks and probabilistic topology changes.

The paper is organised as follows: we give background on RPL and bigraphs in Section 2 and Section 3, respectively. We describe our model for RPL in Section 4, and, in Section 5, we validate it against ContikiRPL and a set of logical properties through model checking. Extensions of the model are in Section 6, and Section 7 discusses the related work. Section 8 concludes the paper and identifies some promising directions for future work.

2 RPL: ROUTING PROTOCOL FOR LOW-POWER AND LOSSY NETWORKS

While in-network computing is possible, a common mode of operation for WSNs is to send all data to a special *gateway* node that often has additional capabilities, *e.g.* long-range radio to upload data to a cloud environment. RPL belongs to the family of distance vector routing protocols, *i.e.* those that use a notion of *distance* (or distance proxy such as latency), to determine the best path for a packet. It is defined by IETF RFC 6550 [3] and implemented for many WSN toolkits, most notably Contiki-NG.

Given a physical network topology, *e.g.* sensor positions and radio ranges, and a predefined *root* node, usually a gateway between the sensor nodes and the Internet, RPL builds a Destination Oriented Directed Acyclic Graph (DODAG) that defines a (directed) routing path from the root to each reachable node. To avoid and detect loops in a routing path, RPL assigns each node with a *rank* that represents the distance from the root.

Computation of a node rank depends on the *objective function* employed by the RPL instance. RPL defines four control messages:

- DIS (DODAG Information Solicitation): allows a node to request information about a nearby RPL DODAG, *e.g.* to determine if it might want to join.
- DIO (DODAG Information Object): transmits information, *e.g.* rank, to other nodes regardless if they have already joined the DODAG.
- DAO (Destination Advertisement Object): is used by nodes joining (or changing position within) the DODAG to update other nodes with new routing information.
- DAO-ACK (Destination Advertisement Object Acknowledgement): is sent by the root to a joining node that, upon receiving it, becomes reachable and can multicast DIOs to other nodes.

We illustrate the protocol by describing informally the steps required to construct a DODAG. Initially, the root multicasts a DIO to advertise the DODAG to any nodes in range. A node that has not already joined the DODAG, on receiving the DIO, can join the DODAG by selecting the sender as its *preferred parent*. If the node is already part of the DODAG, it can either ignore the DIO or select the sender as its new preferred parent if that improves on the current rank. Upon joining, a node unicasts a DAO message to its (preferred) parent. This DAO is further propagated until it reaches the root which replies with a DAO-ACK. After receiving the DAO-ACK, newly joined nodes themselves begin multicasting DIOs to allow for more nodes to join the DODAG through the same process. Nodes not yet in the DODAG

[2]https://docs.contiki-ng.org/en/develop/doc/programming/RPL.html

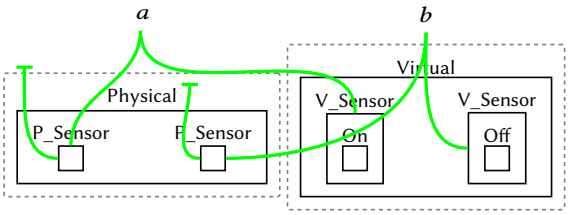


Fig. 1. Modelling a WSN as a bigraph. Left region is *physical* perspective and right region is *virtual*.

may periodically send DIS messages to solicit more DIO messages nodes if required. Eventually, all reachable nodes will be part of the DODAG and the initial route construction is complete.

In this work, we consider the *non-storing* mode of RPL, known as RPL-Lite, *i.e.* routing tables are only maintained at the root and routing information is included in downwards packets[3]. Furthermore, we only focus on the construction of the RPL DODAG, assume a single RPL instance, and use objective function OF0 [49] that assigns uniform weights to links. We will use the abbreviation DAG instead of DODAG for the rest of the paper.

3 BIGRAPHS

Bigraphs are a graph-based modelling formalism, introduced by Milner, to model systems that evolve in both time and space [35]. Bigraphs consists of a pair of relations over the same set of entities: a place graph and a link graph. The place graph models the spatial structure, *e.g.* a sensor *in* a specific room, while the link graph describes non-local connections, *e.g.* communication over a network, regardless of physical location.

We introduce bigraphs by showing a simple model of a sensor network in Fig. 1. Entities, *e.g.* P_Sensor, V_Sensor, are user-defined and may be related spatially through *nesting e.g.* Physical entity contains P_Sensor entity, or (non-local) hyperlinks, *e.g.* connecting a P_Sensor entity to a V_Sensor entity. As links are hyperlinks they connect 1-to-n rather than more traditional 1-to-1 links. Each entity has fixed arity that determines the number of (green) links it must have *e.g.* each P_sensor has two links. Links must always be present but might not connect anywhere (a 1-to-0 link) as shown by an orthogonal line at the end of a link. Links may be named, in which case they are open to extension, *e.g.* might connect elsewhere in some larger model.

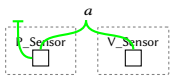
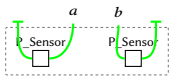
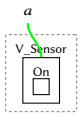
Dashed rectangles, called regions, represent adjacent parts of a system. We use these here to form *perspectives*: different views on the same system. In this case, we decouple the physical aspects of sensors, *e.g.* radio links, to virtual elements, *e.g.* sensor status. Links between the perspectives allow us to track relationships between entities in different perspectives, *e.g.* each physical sensor P_sensor connects to a virtual sensor V_sensor. This multi-perspective approach has been used to good effect to describe mixed reality games [10] and large scale sensor networks [46].

Bigraphs permit an equivalent[4] algebraic and diagrammatic notation. For example, the bigraph in Fig. 1 can be expressed as

$$\mathsf{Physical}.(/c\ \mathsf{P\_Sensor}_{ca} \mid /c\ \mathsf{P\_sensor}_{cb}) \parallel \mathsf{Virtual}.(\mathsf{V\_Sensor}_{a}.\mathsf{On} \mid \mathsf{V\_Sensor}_{b}.\mathsf{Off})$$

[3] An alternative (storing mode) is to have each sensor maintain a routing table.
[4] The equivalence means we lose nothing from the theory by using diagrams over algebra, so this is not a simplification of what a modeller would write.

Table 1. Equivalent diagrammatic and algebraic definition of some example bigraphs.

Operation	Diagrammatic	Algebraic
Parallel product		$/c\ \mathsf{P\_Sensor}_{ca} \parallel \mathsf{V\_Sensor}_{a}$
Merge product		$/c\ \mathsf{P\_Sensor}_{ca} \mid /c\ \mathsf{P\_Sensor}_{cb}$
Nesting		$\mathsf{V\_Sensor}_{a}.\mathsf{On}$
Name closure		$/c\ \mathsf{P\_Sensor}_{ca}$

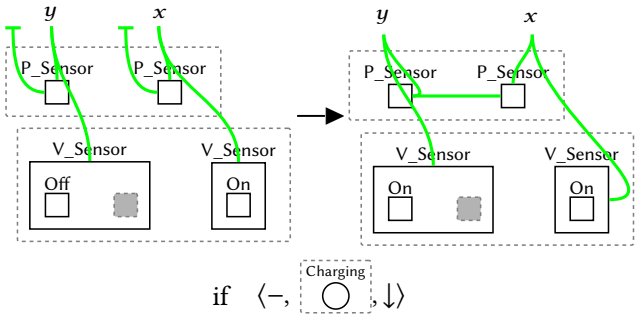


Fig. 2. Example reaction rule: physically connect and turn on sensors that have finished charging.

The equivalence between the algebraic definition of the components and operations used in the example and their diagram form is shown in Table 1. We use the diagrammatic notation in this work as it is easier to understand for non-experts.

For space, we do not give a full overview of Bigraphs here, and refer the reader to [35] for full details. Many variants of bigraphs have been developed, and throughout this paper we use bigraphs to mean conditional bigraphs [5].

3.1 Bigraphical Reactive Systems (BRSs)

Bigraphs specify a system at a particular point in time. To encode dynamics we can equip bigraphs with a set of reaction rules $L \longrightarrow\!\!\!\!\blacktriangleright R$ that states that we can evolve a bigraph by replacing *matches* of a bigraph L with R. A set of all bigraphs closed under the set of rules is known as a bigraphical reactive system (BRS).

An example reaction rule is in Fig. 2 and shows a scenario where two P_Sensor entities can be linked when one sensor is Off and the other On. To allow rewrites to apply in a wide range of circumstances, we can use special entities known as *sites* and shown as the filled dashed grey rectangles. Sites represent elements of the bigraph that have been abstracted away, that is, an unspecified bigraph is allowed to exist there (including the empty bigraphs). This means this single rule matches the case when the V_Sensor might include other information like a sensor identifier. As there is

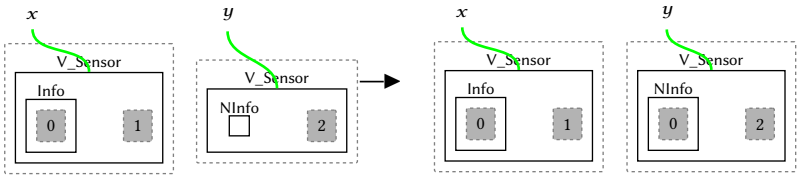


Fig. 3. Instantiation map specified by the numbering on sites. Info is copied to the second sensor NInfo.

no site on the rightmost virtual sensor this is not allowed to nest any other nodes or there is no valid match. Named links act likewise: if there is a name then the link might (or might not) link elsewhere, if not then it can *only* link the entities specified.

In general a rule can apply whenever there is an appropriate match. However, to gain more control over rule application, various extensions to BRSs have been proposed including *rule priorities*—where for two rule sets $\{r_1, r_2\} < \{r_3\}$ none of r_1, r_2 apply unless there is no match for r_3—and *conditional rewriting* that allows rule application only when a given bigraph does/does not appear within a site. For example in Fig. 2 we have the extra condition if $\langle -,$  $, \downarrow \rangle$ that states we do not $(-)$ allow the bigraphs with a single Charging entity in (any of) the sites $(\downarrow)$[5].

To aid modelling, we allow entities and rules to be *parameterised*. This allows generating entities/rules over a set of pre-defined values. For example we can introduce an entity Rank(n) representing a sensors rank. This is syntactic sugar for Rank(1), Rank(2), ... Rules instantiated by a parameter assign that parameter value to entities within the rule.

We use *instantiation maps* to assist in manipulating the content of sites during the application of reaction rules allowing parts of a bigraph to be duplicated or discarded. An example instantion map is in Fig. 3 and shown as numbers inside sites. Sites on the right side of a rule copy the bigraph from the similarly numbered site on the left. For example, this rule copies any bigraph within the Info entity on the left to the NInfo (neighbour information) entity on the right indicated by (site 0) in both sides. If there is no site on the right corresponding to a site on the left, then the contents are deleted.

We employ BigraphER [44], an open-source language and toolkit for bigraphs verification. BigraphER supports rules with instantiation maps, rule priorities, and parameterised entities and rules. Model checking is supported by generating the full transition system that can then be verified by existing tools such as PRISM [31], allowing, for example, temporal logic properties to be checked. To make it easier to write properties, BigraphER allows states to be labelled using bigraph patterns [10] that can label a state whenever a specific bigraph occurs in it.

4 MODELLING RPL WITH BIGRAPHS

In this section, we show how to model the RPL-Lite protocol as a BRS. We begin by describing how we model a given topology and then show how RPL constructs a DAG by adding dynamics.

4.1 Network Topology

There are two communication topologies present in an RPL-enabled WSN. The first is based on the *physical* capabilities of the nodes, *i.e.* the radio range, that determines which nodes may physically communicate. RPL then *overlays* a *routing* topology to provide unicast communication between nodes. That is, although many nodes might be able to communicate physically, RPL is selecting which *should* communicate.

[5]More generally we can write a context condition to force something to appear anywhere not in a site or match, but we do not use this here.

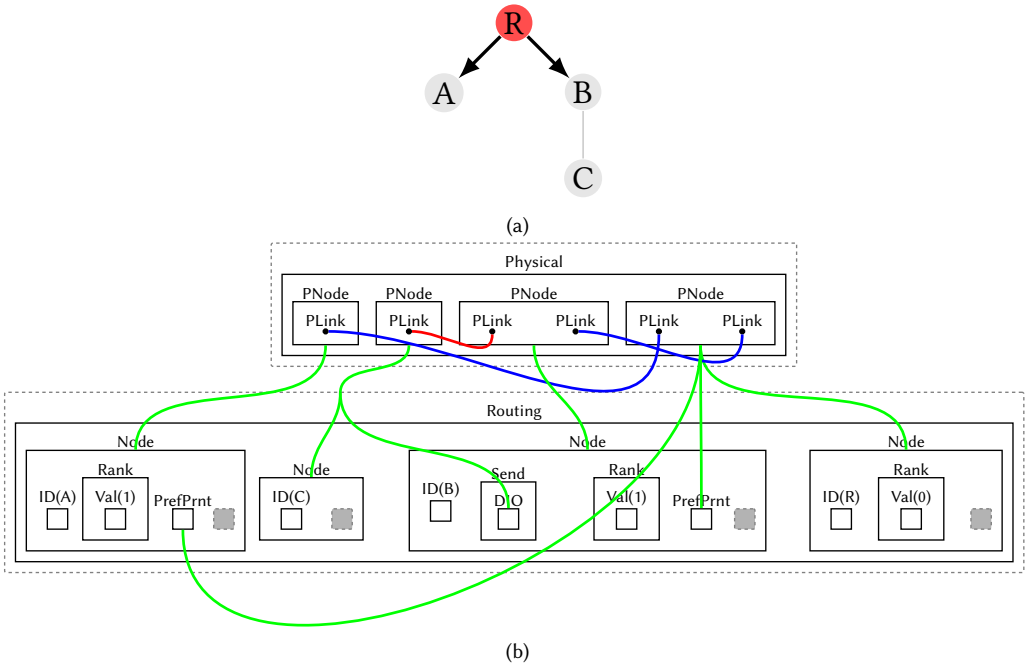


Fig. 4. Example WSN topology (a) and corresponding bigraph representation (b). The partial RPL DAG construction is shown in the physical and routing perspectives.

4.1.1 Example WSN Topology. We use perspectives to capture these two topologies as shown in Fig. 4b. This is a partially constructed DAG indicated by a preferred parent (PrefPrnt) and Rank in Node A and Node B (where nodes are identified by nesting an ID entity). A communication has also started between Node B which is sending a DIO to Node C. Fig. 4a shows a corresponding WSN with four sensors: three connected physically and an additional sensor in range of sensor B only. Black links indicate connected nodes while the grey one is for nodes that are in range but not yet connected. We indicate the root node in red.

4.1.2 Physical Perspective. Models the nodes (PNode) and their physical neighbourhoods, *i.e.* which nodes are in radio range. As we do not model Euclidean distance, nodes are in-range whenever they are connected by linked PLink entities (drawn as small black circles). For clarity, we assume radio ranges are symmetric so pairs of nodes are either in-range or not. The model could be extended to support non-symmetric ranges through additional links, *e.g.* PLinkOut/PLinkIn. We give radio links specific states that change as the model evolves. There are two link states: Idle in blue, and In-use in red. Currently we assume the physical topology is fixed, that is, nodes do not move and the radio ranges do not change due to environmental conditions etc. This could be modelled in future by changing the linking within the perspective (as discussed in Section 6).

4.1.3 Routing Perspective. Contains information about all nodes (Node) and their RPL routing status. Each node has a unique identifier (a parameterised ID entity) and can be assigned a rank (Rank). We identify the gateway/root node as the node with rank 0. Each node, except the root, contains a PrefPrnt entity linked to its preferred parent, for example, Node A and B link to Node R as their parent. Each Node in the routing perspective is connected to the corresponding

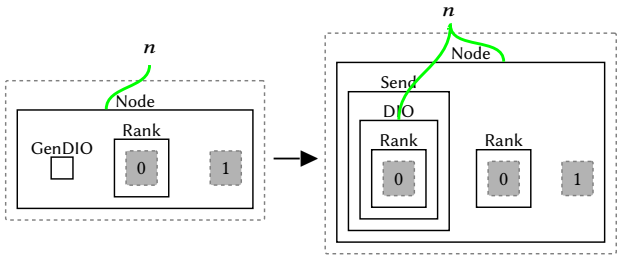


Fig. 5. Reaction rule `generateDIO`.

PNode in the physical perspective via a green link. Control messages are represented by entities DIO and DAO. They are contained within Send or Receive entities in each node to indicate the transmission direction. In practice, more details of the sensors are stored, but we have abstracted these with sites for clarity. These will be introduced as required in future sections.

4.2 Modelling RPL Dynamics

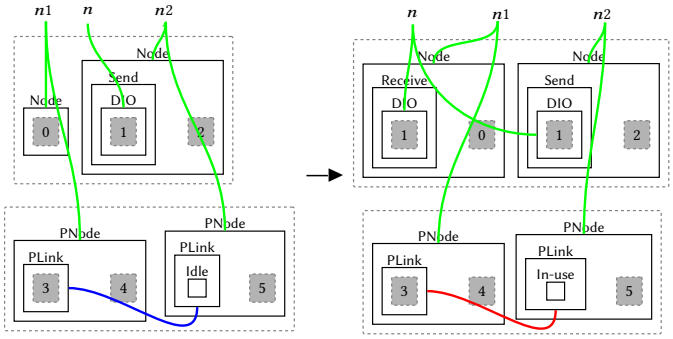
We now show the reaction rules required to model the RPL protocol dynamics, *i.e.* messages between sensors, and sensor local updates. We use the RPL process described in Section 2 as our starting point with the following assumptions:

(1) We only model downward connection where newly joined nodes multicast DIOs to their neighbours. We do not model DIS messages, that let a node request information upwards, as these require clocks, *e.g.* after n seconds try another DIS.
(2) We assume nodes receive one DIO at a time, *i.e.* that the processing time is short enough relative to message transmission time, and assume no packet drops in the network. Note that, because we do full model checking, all possible execution traces, *i.e.* message orders, are considered even if two or more messages appear at the same time.
(3) No node failures occur during DAG construction.

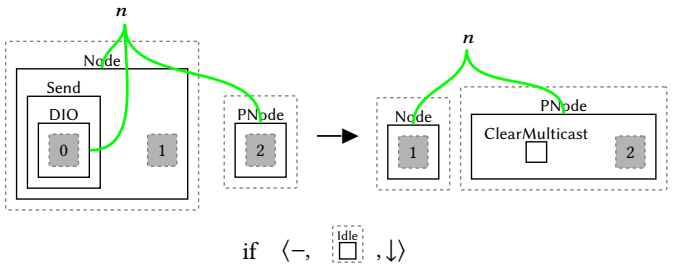
These assumptions keep the model small to allow efficient verification, and to ensure it is succinct enough to be described in the paper, but have no effects on the construction of valid DAGs. We can remove these assumptions by adding additional reaction rules like we show in Section 6. Checking of specific timings (rather than the symbolic version that captures all interleavings that we show here) would require extensions to the bigraph theory, *i.e.* introducing timers/real-time bigraphs.

Our model consists of 16 reaction rules (including two parameterised rules) encoding the four main steps of the RPL DAG constructions. To keep accurate to the distributed nature of the network, rules rely on updates based on local information only. We give the diagrammatic versions of the reaction rules here, and the algebraic forms are in Appendix A.

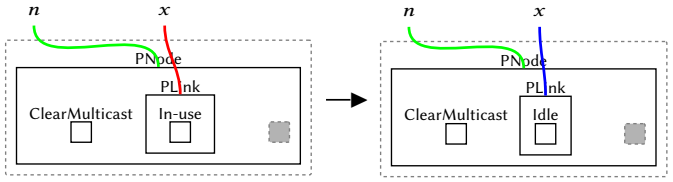
Step 1: Generating DIOs (Fig. 5). The first step is that a new node generates a DIO message to be sent to all its physical neighbours. We use a special token genDIO within a Node to determine when a new DIO should be generated and this is initialised on the root. Rule `generateDIO` converts this to a DIO message represented by a DIO entity within a Send entity. The DIO message contains two bits of information: a *link* to the sending node (n) and a copy (via an



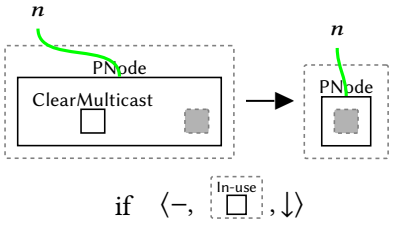
(a) sendDIO



(b) sendDIO_Done



(c) clearMulticast



(d) clearMulticastEnd

Fig. 6. Reaction rules for DIO multicasting process.

instantiation map) of the Rank *at the time the message was sent*[6]. The link to the sending node allows it to be identified, without needing to explicitly track IP addresses (as would be sent in an implementation).

[6]We cannot rely on matching via the link to the sender here, as the rank might update after the message is sent but before it is received.

Step 2: Multicasting DIOs (Fig. 6). Once we have a message to send, the multicast process can start. As we want to send a message to all physical neighbours at once, we utilise a multi-step process. First, rule `sendDIO` (Fig. 6a) determines if there is a physical neighbour that has not yet received the DIO, *i.e.* its link is Idle (blue). If so, it performs the send by coping the DIO message to the neighbour Node, and tags this link as In-use (red) so we do not send twice. The instantiation map allows this rule to be used regardless of the contents of the DIO, *e.g.* the specific rank being sent. A second rule `sendDIO_Done` (Fig. 6b) removes the Send entity and its content once we detect all sends have been completed. This is achieved by specifying a condition checking that there are no Idle physical links. Rule `clearMulticast` (Fig. 6c) reverts the link used for multicasting into its initial state (Idle) while rule `clearMulticastEnd` (Fig. 6d) removes ClearMulticast from PNode entities with Idle links, thus enabling other DIO messages to be sent.

An optimisation is possible using instantaneous rules—an advanced feature of BigraphER—that allow us to apply a set of rules (to a fix-point) as if it was a single rule application. That is, we perform the entire multicasting process as a single step to mimic the physical (radio broadcast) nature.

When two or more nodes have messages to multicast, this will happen around the same time in the model. This is based on the assumption that processing messages (see step 3) is significantly faster than message sending so processing occurs with higher priority in the model leaving message sending rules to be handled together.

Step 3: Handling the DIO (Fig. 7). Once a DIO is received, nodes determine how to handle the message. There are three cases for the receiver: (1) not yet joined, (2) joined and the sender has rank *equal to* or *greater than* the receiver, and (3) joined and the sender has rank *less than* the receiver.

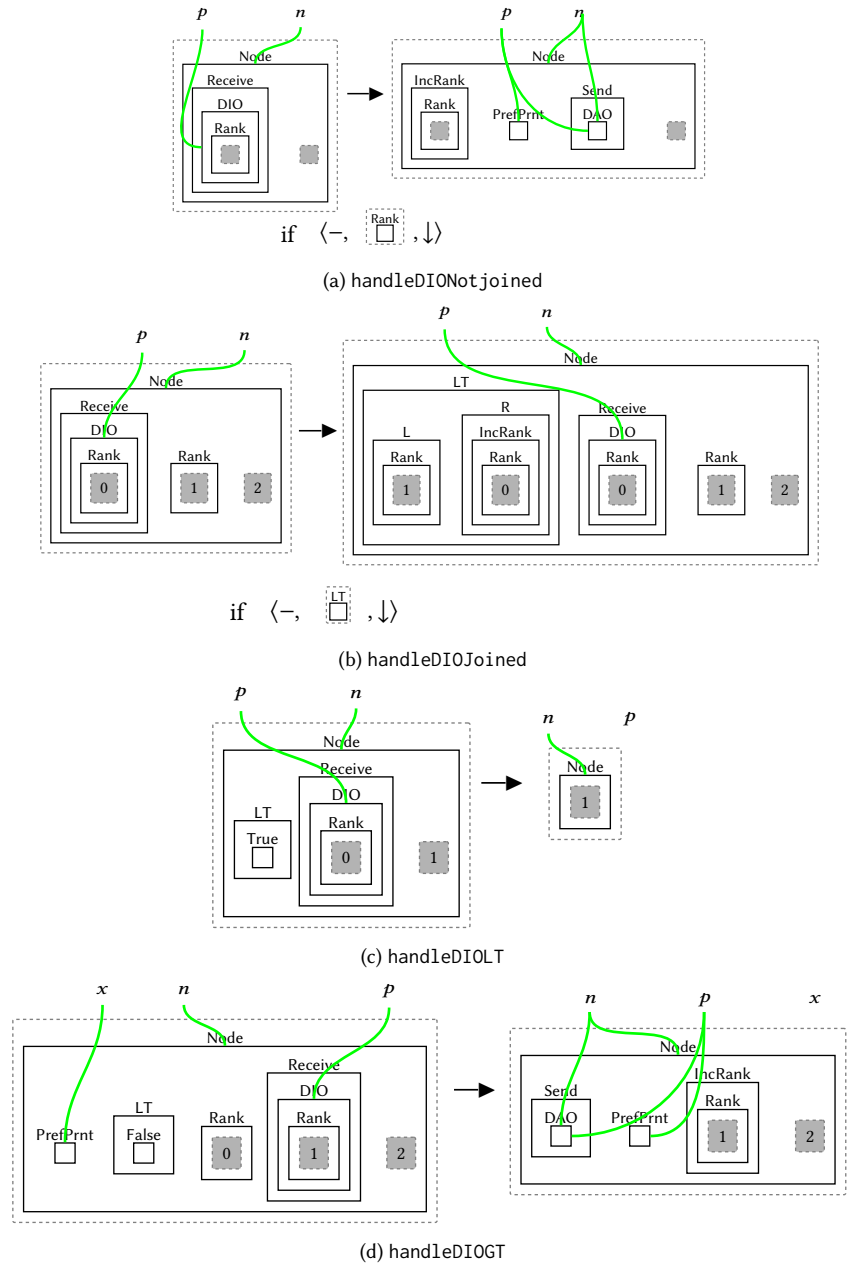
Rule `handleDIONotJoined` (Fig. 7a) handles case (1) knowing the receiver has not already joined by the absence of a Rank (handled in the condition). When the rule applies, the receiver requests the Rank to be the sender rank + 1[7], via IncRank, and sets the preferred parent to be the sender (identified by link p). Finally, a DAO message is scheduled to be sent to the sender by creating a DAO entity linked to p and placing it inside a Send entity.

To allow rank comparisons to be generic, and as bigraphs have no built-in mathematical comparisons, we use the rules in Fig. 8 to perform rank comparisons and increments where n is the current rank for the receiver and m is the potential new rank. The comparison functions follow the standard recursive definition of less-than via repeated subtraction. We use True and False entities (Fig. 8c and Fig. 8d) as flags for the comparison result to be used in the next step; *i.e.* True indicates that no rank update is required.

Rule `handleDIOJoined` (Fig. 7b) handles case (2) when the node is already present in the DAG (*i.e.* it has a rank). Since sender has rank equal to or bigger than the received rank then rule `handleDIOLT` (Fig. 7c) applies and the receiver drops the message (and cleans up the comparison) as it is already in an equal or a better position than if the sender was made a new parent. If the sender has rank less than the receiver (case 3), then `handleDIOGT` (Fig. 7d) applies to allow the receiver to set a new preferred parent and update to this improved rank.

Step 4: Sending the DAO (Fig. 9). Nodes register themselves by sending a DAO to the root. This is achieved by first applying the the rule `sendDAO` (Fig. 9a). If the receiving node is not the root, rule `sendDAOUpParent` is then applied to pass the DAO further up towards the root through the node's preferred parents set (Fig. 9b). `clearDAO` (Fig. 9c) then clears the DAO once it reaches the root. Once the DAO is initially sent, a GenDIO is generated so the receiver will generate new DIO messages (continuing from Step 1).

[7] As required by OF0.



(a) handleDIONotjoined

(b) handleDIOJoined

(c) handleDIOLT

(d) handleDIOGT

Fig. 7. Reaction rules for handling the DIO.

In practice this DIO would only be generated once a DAO-ACK is received from the root node. Since we are not modelling link failure, DAO-ACKs will always be successful so we skip the steps to send them. Modelling DAO-ACKs would be similar to `sendDAOUpParent` if required in future.

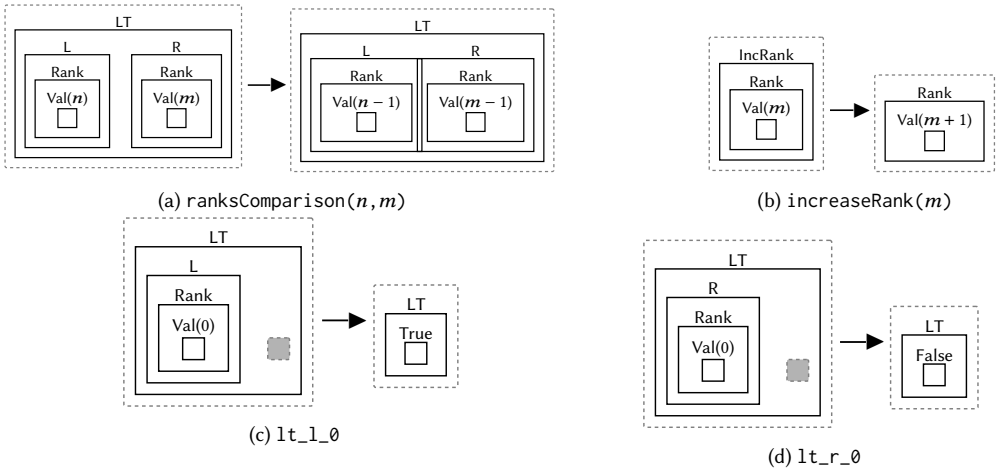


Fig. 8. Reaction rules for rank calculation and comparison.

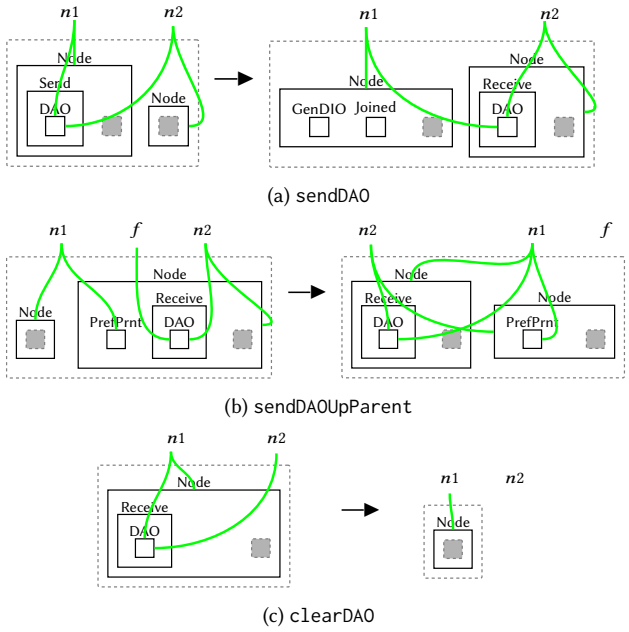


Fig. 9. Reaction rules for sending the DAO.

Once all nodes have joined and fixed their ranks no new DIO messages will be generated. At this stage the DAG construction is complete and all nodes will have joined on a minimum rank possible. The full model including the priorities classes employed to enforce the desired sequence of execution is available at [43].

5 MODEL VALIDATION AND COMPARISON TO SIMULATION APPROACHES

We compare our model against RPL-Lite, the default Contiki-NG's RPL implementation. We use Cooja, which is a popular emulator for IEEE 802.15.4 networks with devices running Contiki-based firmware, to simulate the radio environment.

Our modelling assumptions (listed in Section 4.2) are appropriate when comparing against RPL-Lite as:

(1) The timing aspects (*i.e.* clocks) of RPL-Lite are not considered as they do not play any role in the construction of the initial DAG which is the focus of this work.
(2) Nodes in Cooja can send DIS messages periodically which is not the case in our model. DIS messages are mainly used for DAG repair which we do not currently focus on. The presence of DIS does not enable different DAGs to be constructed than DIO only.
(3) Parent selection in Cooja uses multiple probings to test link quality before selecting a parent (if there are multiple similarly ranked options). As the bigraph model considers all orderings, we will get all possible parents regardless of link quality or the metric employed. One way to view this is that we consider all possible link quality alternatives.

To formally check we get correct DAGs, we specify a set of properties to verify our RPL model through PRISM, an open-source automated verification tool providing model checking for different types of probabilistic models such as Discrete Time Markov Chains (DTMCs). Although our model is non-probabilistic, it is still supported by PRISM by treating the transition systems generated by BigraphER as a DTMC with uniform weight on transitions and utilising only the non-probabilistic fragment of the property logic, *e.g.* Computation Tree Logic (CTL) [17]. The choice of PRISM is motivated by technical reasons, *i.e.* BigraphER currently only supports PRISM output format, but could also be a benefit in future to allow probabilistic modelling within RPL and other protocols, *e.g.* assigning probabilities/rates that some link fails (as shown in Section 6).

We verify RPL features using properties expressed in CTL. CTL uses quantifiers over paths, **A** (for all) and **E** (there exists), and path formulae, **F** (eventually) and **X** (next). For example, **E** [**F** ϕ] means: there exists a path that eventually reaches a state satisfying ϕ (in our case, ϕ is specified using a bigraph pattern).

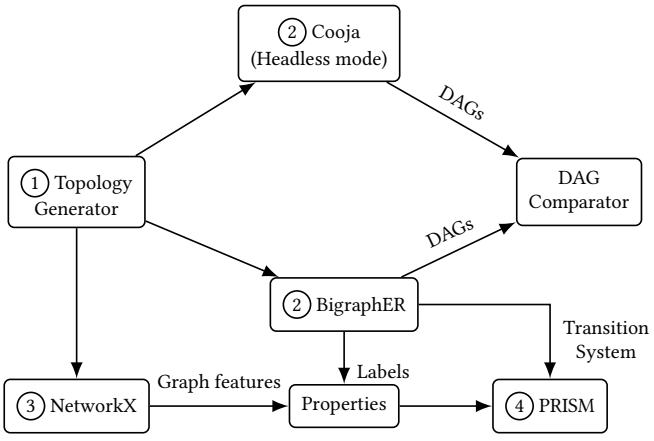


Fig. 10. Experiment approach.

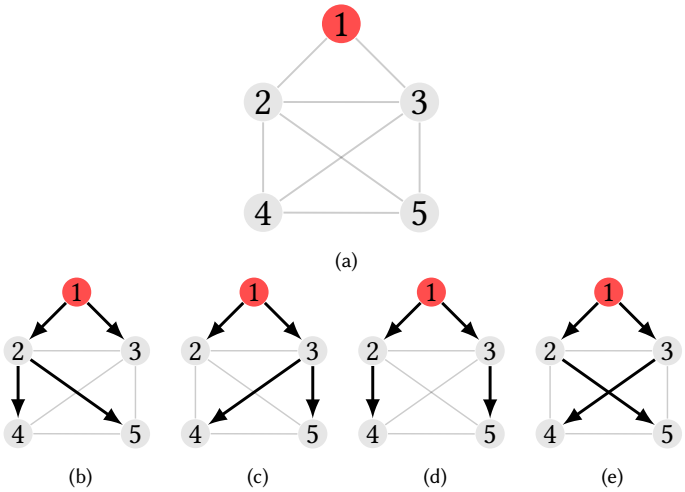


Fig. 11. Example physical topology (a) and all valid RPL DAGs (b)-(e).

5.1 Experiment Setup

To validate our model and allow empirical comparisons with Cooja, we experiment with 100 randomly generated network topologies with between 7 and 9 nodes. Each node is assigned a (uniform) random physical 2D-position (as needed by Cooja), and links for the bigraph models are determined using the position and the wireless transmission range. We only generate fully connected topologies *i.e.* there will always be a path to all nodes, with no self-loops. Code to generate the topologies and to reproduce the experiments is available online [43]. Fig. 10 shows the experiment flow for *each* topology, an example of a five node topology is shown in Fig. 11a. For each topology we do the following:

(1) Encode the generated topology for both Cooja (`.csc` file) and BigraphER (`.big` file) as described in Section 4.1.
(2) Run the model and simulation and extract the DAGs constructed by both BigraphER and Cooja. For each we check if the two DAG sets are the same (often Cooja misses possible DAGs which BigraphER finds as discussed in Section 5.2). The DAGs generated by BigraphER are obtained by parsing all the deadlock states in the transition system while those found by Cooja are constructed by parsing each simulation log file.
(3) We use NetworkX[8], a Python package for the creation, manipulation, and analysis of graphs, to generate a list of predefined properties (*e.g.* the generated DAGs have no cycles) for the validation of the bigraph model.
(4) Verify with PRISM each of the properties in (3) against each DAG constructed by our model. The model checking results are saved in a text file. We assume the DAGs found by Cooja are correct therefore we do not validate them.

We performed our experiments using GNU Parallel [47] on machines with dual 8-core Intel Xeon E5-2640v2 CPUs (2Ghz; no hyper-threading) and 64GB of RAM. We use BigraphER version 2.0.0 and run Cooja on Docker version 24.0.2. In Cooja, we use node type `Z1` with `UDP-server` running on the root and `UDP-client` on the other nodes. We set the transmission range to 100 m, the interference range to 140 m, and use OF0 as the Objective Function. We generate random seeds for each simulation which then runs at 1000x simulation speed and terminates when the simulator constructs a DAG comprising all n nodes.

[8]https://networkx.org/

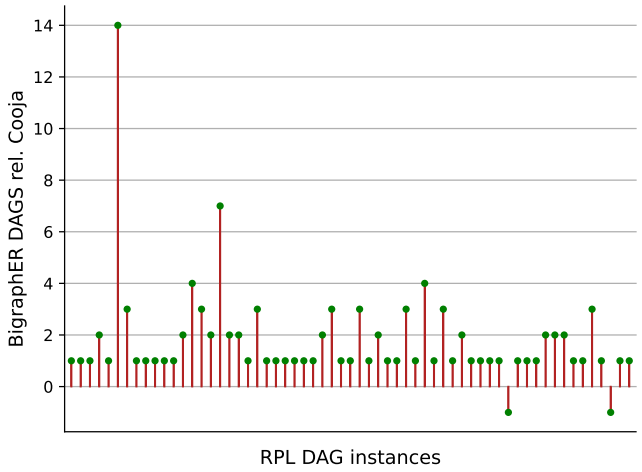


Fig. 12. Number of RPL DAGs found with BigraphER and Cooja: above 0 implies BigraphER found *n more* DAGs than Cooja. Topologies where BigraphER and Cooja agree are unshown (39% of cases).

5.2 Comparing Bigraphs to Cooja

In this section we use our experimental setup to compare the full model checking approach of bigraphs to the simulation approach of Cooja. We use a single run of BigraphER (since this is enough to discover all possible DAGs) and 500 runs of Cooja for each random topology.

5.2.1 Comparing DAGs Found. A key benefit of the model checking approach of bigraphs is that a single run will find all valid RPL DAGs, while a single run of Cooja will find only one. 100 topologies were sufficient to explore our hypothesis that the formal approach would find more valid DAGs than Cooja and provide consistent outcomes. More topologies can be explored using our tooling [43]. Fig. 12 shows the difference in the number of DAGs found between BigraphER and Cooja (after 500 simulation). BigraphER finds more DAGs than Cooja in more than half of all cases (59/100), and in some instances these can be quite startling: in one instance BigraphER detecting 14 more valid DAGs than Cooja. Even increasing this case to 1000 runs finds no more DAGs.

There are two cases where BigraphER times out (> 24 hours runtime). Since we do not currently support partial output that shows us how many DAGs are discovered before this happens, we utilise BigraphER simulation to support our analysis. In one case, BigraphER finds all possible DAGs *i.e.* 4 DAGs in 9 runs while Cooja finds only 1 in 500 runs. In the other case, there is only one possible DAG that both BigraphER and Cooja find. We chose 500 simulation runs in the hope Cooja had enough time to find interesting RPL DAGs.

In Fig. 13 we show, for all instances where 3 or 4 RPL DAGs were found, how likely *each* different DAG was to be found. These results show Cooja usually prefers a single RPL DAG that it finds in more than 70% of runs; often in 80% of runs. Finding a third or fourth DAG happens in only a very small number of cases (a fifth DAG was found in one instance only) and it is very sensitive to the number of simulations performed. That is, truly exploring RPL DAGs with Cooja is likely to require a large amount of computational resource if we want to be confident of finding all cases. Bigraphs do not have this limitation.

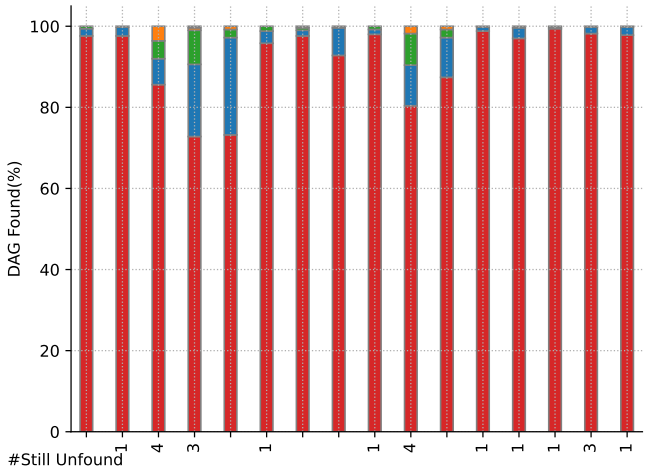


Fig. 13. Distribution of RPL DAGs within Cooja simulations (for the 16 instances that find 3 or 4 DAGs). Colours indicate the different DAGs found; *i.e.* red colour indicates the most DAG repeatedly found in 500 simulations. x-labels show instances where more DAGs were possible, but never discovered even after 500 simulations.

The differences between Cooja and BigraphER DAGs has implications for future protocol analysis: it is clear neither is sufficient on its own. Full model checking can show many more cases to increase confidence the protocols work in a much wider range of environments, but in a small number of instances state space explosion issues may make it impractical to use and simulation can still provide insights in these cases.

5.2.2 Running Times. We have shown BigraphER finds more valid RPL DAGs than Cooja in most cases. As Fig. 12 indicates, Cooja finds the same number of valid RPL DAGs as BigraphER does in only 39% of cases, and this highlights the issues of simulation missing outputs when analysing protocols.

There are trade-offs in the time required to discover all DAGs (bigraphs) instead of a simulated subset of DAGs (Cooja). Fig. 14 shows the distribution of BigraphER and Cooja runtimes. BigraphER runtimes are clustered towards the left, showing that in most cases (83/100) we can find all RPL DAGs in less than 60 minutes. Of the remaining instances, most complete within 180 minutes (3 hours), but 2 take longer than 1 day to compute (so time out). Due to the combinatorial nature of these problems it is not possible to predict a-priori how long a particular run will take.

Cooja runtimes are much less spread, and take a mean runtime (for 500 simulations) of 154 minutes (2.5 hours). Given the increased number of DAGs found, and the fact BigraphER usually manages to find these in less than 1 hour, the modelling approach seems to be a valid alternative to Cooja, but, as we mentioned, we believe the best approach is for both techniques to be used in tandem.

5.3 Verification

We consider two approaches to verification: model checking (over a given topology) and static analysis of the reaction rules to prove properties hold regardless of the specific topology. Model checking is useful for small toplogies due to the state-space explosion issue which is common in formal verification but benefits from the automated nature. In future, the support for probabilistic reasoning can also be utilised, *e.g.* to model real link failure rates.

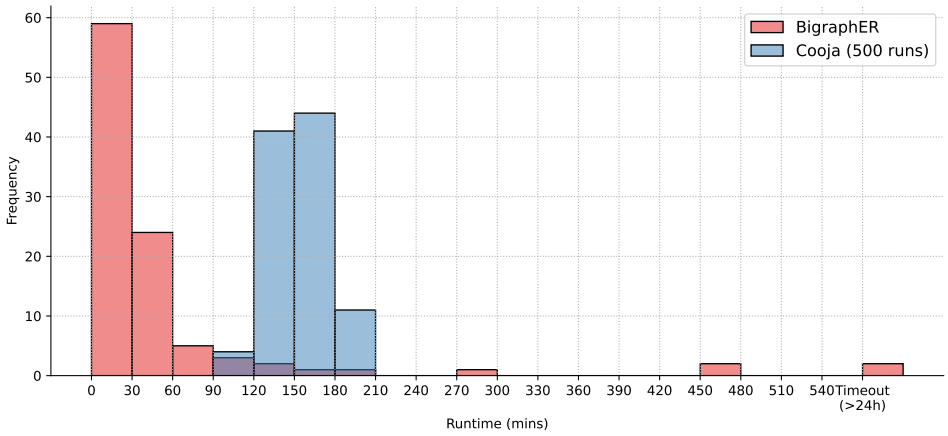


Fig. 14. Distribution of BigraphER and Cooja runtimes. Timeout is 24 hours.

5.3.1 Model checking. To check all the generated DAGs are valid, we verify the bigraph model using the PRISM model checker. We use NetworkX to assist in generating a set of properties to validate the model. This because the current definition of bigraph patterns does not include spatial modalities like for instance $\mathcal{R}$ (*reachable from*) in SLCS [15, 32] that would allow reasoning on path properties on the place graph. Here, we employ NetworkX to automatically extract the required graph properties (*e.g.* the shortest path between two nodes) from each DAG constructed by the bigraph model and include them into a set of CTL formulae in PRISM format. In essence, this allows us to check that our local protocol creates correct global DAGs.

We check the following properties for each run. These are temporal properties so these check that they hold for all possible paths in the transition system. In the following we use n to indicate the number of nodes in a topology.

Property 1: *All nodes eventually join the RPL DAG.* As we assume fully connected physical topologies, RPL should eventually find a path between the root and all nodes. We check this by assigning a label $\texttt{joined}_i$ whenever a node i is assigned a rank and so is part of the DAG (Fig. 15a).

To check all nodes eventually join, we use the following *family* (*i.e.* one property per i[9]) of properties:

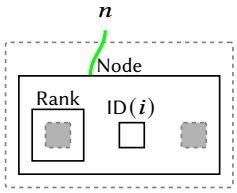
$$\forall i.\ \mathbf{A}\,[\,\mathbf{F}\,\texttt{joined}_i\,] \quad \text{where } 1 \leq i \leq n \tag{1}$$

This states there is *always* a path such that node i *eventually* joins.

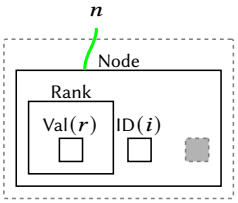
Property 2: *Nodes join with the optimal rank.* RPL is designed to find optimal routes based on minimising the objective function. In general there may be many optimal routes, but they will always lead to a node having the same rank, *e.g.* in Fig. 11b, and Fig. 11c node 4 will always eventually have rank 2 even though there are (at least) two different valid routes to it. Here we check nodes eventually join on this optimal rank.

To write this property, we get the length of the shortest path from each node i to the root with NetworkX's function $sp(i)$, and use $r = sp(i)$ to denote the rank a node i would be assigned on this path. We also add a label $\texttt{rank}_{i,r}$ that occurs whenever a node i is assigned rank r (Fig. 15b).

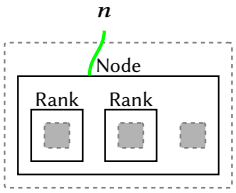
[9]To ensure we can enumerate these properties we ensure i etc. is drawn from a finite set of identifiers.



(a) $\texttt{joined}_i$: node i joined the RPL DAG.



(b) $\texttt{joined}_{i,r}$: node i joined with rank r.



(c) `multijoin`: a node joins at least twice. This should never happen.

Fig. 15. Bigraphs predicates to check RPL properties.

To check each node joins with optimal rank we then use the (family of) property:

$$\forall i.\mathbf{A}\left[\mathbf{F}\,\texttt{rank}_{i,r}\right] \quad \text{where } 1 \leq i \leq n,\ r = sp(i) \tag{2}$$

As this checks all paths, this works even if a node temporarily joins with the "wrong" rank, *e.g.* if it has found *a* route, but not yet the optimal route.

Property 3: *RPL DAG is cycle-free.* To ensure messages do not get stuck in infinite cycles, we want to check the DAG create by RPL is always actually acyclic. The mechanism RPL uses to achieve this is the ranks, and we can guarantee cycle-freedom by ensuring nodes only ever join on a single rank (else a node could be a descendent of itself).

We do this using a labelling `multijoin` that matches using the bigraph shown in Fig. 15c, *i.e.* two ranks and anything else. We can then check this *never* occurs:

$$\mathbf{A}\,[\,\mathbf{G}\,\neg\texttt{multijoin}\,] \tag{3}$$

5.3.2 Static analysis. While the model checking approach is useful as it can automatically find property violations, to show properties of more general topologies it is possible to use a (non-automated) static analysis approach. Our strategy is to show that, regardless of the specific topology, the following properties must hold under application of all possible (and applicable) reaction rules. Proper execution of the rules is guaranteed by using priority classes that provide an ordering on the rules. We use an inductive argument to prove the first property and invariant reasoning for the second one. In both cases we assume a connected topology.

Property 4: *All nodes eventually join the RPL DAG.* Note this was proved for a given topology with model checking in Property 1.

For the base case, we show that any child of the root that is not already in the DAG, eventually joins it, *i.e.* the child node gets a rank. The only applicable rule is `generateDIO` as, by construction, the only node with a GenDIO is initially the root. This will introduce a DIO in the root enabling sendDIO, thus triggering the multicast process

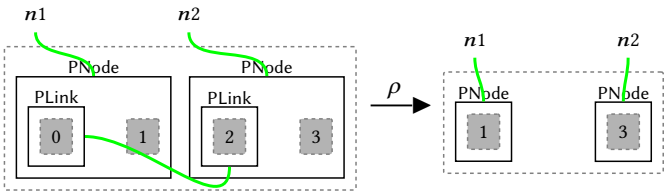


Fig. 16. Stochastic reaction rule dropLink with rate ρ.

described in Fig. 6 terminating with an application of rule `clearMulticastEnd`. Although different interleavings are possible as children can be processed in different order, this step of rewriting is confluent. Since we assume the children were not already in the DAG, the only applicable rule is now `handleDIONotjoined` followed by `incRank(0)`. Different interleavings of these two rewriting steps are possible, but eventually all the children will have a Rank. That allows `sendDAO` to be applied, thus introducing a Joined entity in every child.

For the inductive step, it is sufficient to show that any node that has a parent in the DAG but is not already in the DAG, eventually joins it. Any parent already in the DAG acts as the root in the base case and the same sequence of rewriting can be applied with two differences:

- other rules from Fig. 7 (DIO handling) and Fig. 8 (rank computations) can be interleaved as the parent node can also transmit to connected nodes already in the DAG;
- rule `incRank`(n) with $n > 0$ is applied for new nodes.

Property 5: *RPL DAG is cycle-free.* This is the static equivalent of Property 3. Since by construction, in any initial state only the root has one Rank and pattern `multijoin` in Fig. 15c never occurs in the right-hand side of any reaction rule in our model, then we can never reach a state in which two Rank entities are within a Node at the same level of nesting.

6 RPL MODEL EXTENSION

The bigraph model is flexible and open to extension, without requiring full implementation (*e.g.* of Contiki modules), by adding additional reaction rules as required. Our model can be seen as a base of RPL formal description that not only models its core element *i.e.* DAGs but can also be extended to include more functionalities, such as hardware and communication failures and RPL attacks. As an example, we show how our model can be extended to include physical links drops and a common RPL attack *i.e.* sinkhole attack [52, 26], but leave a full analysis as future work.

Physical links drops. We show how to extend the bigraph model to include variability in the radio signal due to interference or other environmental factors. We add a *stochastic* reaction rule [29, 6], as in Fig. 16, that drops the physical link between two nodes with rate $\rho \in \mathbb{R}_{>0}$. When this event gets triggered, it forces the RPL protocol to construct a DAG on a new topology[10]. Quantitative analysis on the extended model can be carried out with PRISM by expressing the properties in Section 5.3 using Continuous Stochastic Logic (CSL) [8] instead of CTL. For example, property $\mathcal{P}_{\geq p}$ [F $\texttt{joined}_i$] holds if node i (with $1 \leq i \leq n$) eventually joins the RPL DAG with probability greater or equal than p.

[10]This requires modelling trickle-timers for repair steps.

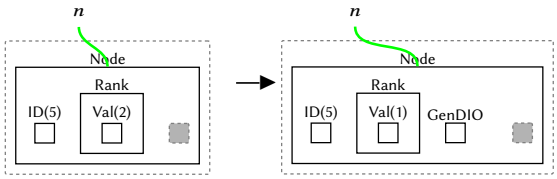


Fig. 17. Reaction rule `sinkholeAttack`: node 5 advertises a fake rank.

Sinkhole Attack. In this scenario, a compromised node advertises a fake rank to get neighbours to connect to it. We model this by first assuming a reaction rule (not shown) that randomly selects a node, other than the root, to be a compromised node. Then use a reaction rule as in Fig. 17 that gives a (randomly selected) node (5 in the figure) rank 1 and requests for a DIO to be sent by generating a GenDIO in the right-hand side. When other nodes receive the DIO they will update their rank accordingly, assuming there is a better path to the root. A second custom rule (not shown) for compromised nodes would stop them forwarding messages, *i.e.* they become sinks. We assume the reaction rule `sinkholeAttack` has the lowest priority so that they only take place once the RPL DAG is constructed. This could also be achieved using a second model that takes the result from the RPL construction stage. Ideally, all results would be used to ensure full coverage, but this is potentially place to mix simulation and modelling, *e.g.* let Cooja generate the most common topology then exhaustively check security via bigraphs.

7 RELATED WORK

Applying formal approaches to the design and analysis of IoT protocols is an established research field. Like ourselves, several works emphasise the significance of utilising protocol analysis for protocol design.

Glabbeek et al. [22] provide a formal specification of the main functionality of an Ad-hoc On-demand Distance Vector (AODV) routing protocol, of which RPL is an instance, using Algebra of Wireless Networks (AWN). Similar to our model, the AODV model covers the main components of the protocol while abstracting timing aspects and optional features. The work reasons about key features of AODV *i.e.* loop freedom and route correctness which are used as examples. The authors believe, as we do, that other protocol evaluation approaches such as simulation and testbed experiments, do not provide a robust answer regarding protocol behaviour due to their limited coverage of network scenarios. They claim that using model checkers, such as UPPAAL, ensures discovering potential errors in an early protocol development stage. They state that due to the use of English prose in specifying AODV specification, it holds ambiguity that may affect implementations and they also suspect that might be the case for other IETF specifications.

Many secured routing protocols for ad-hoc networks contain design flaws hence they are vulnerable to attacks as claimed by [48]. The author uses the deductive proof technique and a backward reachability approach to provide an analysis framework to automatically verify the correctness of secure ad-hoc network routing protocols. They develop a software tool based on their proposed approach that finds attack scenarios successfully in well-known routing protocols.

To verify that the synchronisation and dissemination protocol for Wireless Sensor Networks satisfied its requirements, [53] converts the protocol into a logical model to reason about it using PRISM model checker and probabilistic computation tree logic (PCTL). The employment of probabilistic models permits the conducting of a quantitative analysis of protocols under the effects of environmental changes. Our analysis shows an agreement with what the authors conclude which is that the utilising of multiple methods to verify critical IoT systems should be taken into account as they usually possess complementary characteristics.

Existing modelling tools can be readily applied to protocol analysis. Coloured Petri Nets (CPNs) [25] are used to model and verify MQTT protocol logic that covers all three quality of service levels[11] which MQTT provides for message delivery [40]. They employ both simulations, to test the appropriate operation of the model, and exhaustive validation by exploring the state space. This resulted in finding several issues regarding the implementation of the quality of service levels which may lead to interoperability problems between implementations. They manage the state space explosion by employing an incremental model checking approach based on the different stages of the MQTT protocol. In addition, they impose some assumptions to restrict the state space *i.e.* assuming a limited number of clients and considering a single topic, also bounded packet identifiers.

The Ad-hoc On-demand Distance Vector (AODV) routing protocol has been modelled [33] using Maude [18], a rewriting engine that provides a variety of analysis techniques such as LTL model-checking. The model does not consider message collision to avoid the state space explosion. The authors conclude by suggesting that statistical model checking techniques and/or abstraction techniques for Mobile ad hoc networks (MANETs) should be developed. Alloy [24], a modelling and analysis language, is utilised to model the Android permission protocol and provides a fully automatic analysis that leads to identifying three types of vulnerabilities that may permit unauthorised access [9].

Other approaches rely on verification tools that specifically designed to support protocol design and analysis. For example, ProVerif [11] is used to verify proposed enhancements to an existing WSNs authentication protocol that suffers from some attacks and violates secrecy and anonymity [55]. However, [21] utilises ProVerif to simulate and verify an enhanced symmetric key-based authentication protocol for IoT-based WSNs. Tamarin [34] is another widely used tool for protocol verification, especially for security properties. It is employed to verify a proposal for enhancing the Reactive-Greedy-Reactive (RGR) routing protocol for Unmanned Aeronautical Ad-hoc Networks (UAANETs) [37]. In [20], it is used to verify a proposed formal definition of Flow Integrity which the authors applied to industrial systems. While these tools do not suffer from state space explosion as they use an abstraction on protocols with unbounded sessions, this abstraction may lead to reporting false attacks when analysing protocols with global states. Another limitation is that they lose automation as supporting protocols with no limitations of states requires human intervention by providing auxiliary lemmata. This can be challenging even for experts [56].

Despite the popularity of the RPL protocol, there are few works formally modelling it. A recent paper [1] employs coloured Petri nets (CPNs) to model the security schemes of a subset of RPL covering single instance and *pre-installed mode*[12]. The authors model RPL control messages—focusing on the DIO and DAO messages as they enable interaction with the other nodes—and evaluate the existing security standards for RPL through state-space analysis. When there is no loop in the network topology, their model satisfies the desired state. However, since they do not implement loop discovery and inconsistency repair, the model may provide undesired results that are caused by incorrect topologies.

Regarding bigraphs, they have modelled a wide range of networking protocols and systems. For example, self-adaptive Fog systems which consist of distributed micro data-centres in the core network and resource-scarce devices at the Edge of the network [42]. A large-scale WSN (up to 200 nodes) is also modelled by bigraphs to enable reasoning about the spatial, operational and behavioural aspects. The authors use BigraphER to automate online verification, while Cooja is used to generate events and data streams to allow reply of realistic network events and sensed data [46]. The structure and behaviour of Wireless Mesh Networks (WMNs) were modelled as bigraphs together with Maude for analysis [13]. Bigraphs with sharing [45] allows entities to have multiple parents, and allowed modelling the 802.11 CSMA/CA RTS/CTS protocol with signal overlaps directly specified [14].

[11] MQTT provides three quality of service levels for delivering messages between clients and servers: *at most once*, *at least once*, and *exactly once*.
[12] In this mode, a node should have already installed a cryptographic key on it to be able to join an RPL instance.

8 CONCLUSION AND FUTURE WORK

Formal models allow detailed reasoning about the correctness of networking protocols. Milner's Bigraphs provide a modelling framework that is diagrammatic in nature making it usable by a wide audience. We have shown how bigraphs, using only 24 entities and 16 reaction rules, can model the initial routing DAG construction step of RPL—a popular protocol for wireless sensor networks. A key benefit of the formal approach is that we obtain all possible DAG construction paths within a single run, and although this can take a significant time, this allows us to check the protocol works in a much larger set of situations (in one case with 14 more DAGs) than is found using emulation in Cooja. Bigraphs are open to extension by simply adding a few additional reaction rules, rather than requiring a full implementation effort *e.g.* as would be required to implement changes for Cooja. We have shown how some security issues could be modelled, and a future step would be to prove existing mitigation, *e.g.* local-repair steps, are sufficient to counteract attacks including situations where multiple attack types occur in parallel. We believe a mix of both formal modelling and emulation/simulation should be considered for future protocol design, with emulation allowing experimentation at a much larger scale while modelling is a quick way to try new ideas while giving robust answers, *e.g.* for all possible DAGs. This robustness is essential: even if we only see a DAG a very low percentage of the time, it is much better we have tested for it ahead of time than assume it will never happen.

As future work we wish to explore more details of the RPL protocol, specifically the trickle timing mechanism for DAG repair. Currently bigraphs do not directly support timers, however some preliminary work [2] shows it is possible to encode digital clocks by leveraging the Markov Decision Processes semantics provided by *action bigraphs* [6].

We would also like to drop some assumptions, *e.g.* that each node receives only one DIO at a time by adding message queues, and capture a larger proportion of the RPL protocol.

We believe that other (distance vector) routing protocols, *e.g.* Thread [51] or LoRaWAN [4], can be modelled using a similar approach by utilising common features between protocols. For example, routing in Thread similarly has routers periodically send their routing 'costs' to all other routers, which is similar to the DIO broadcasting of RPL. We believe modelling multiple protocols could beneficially determine a set of common approaches for modelling protocols in general.

ACKNOWLEDGEMENT

We thank Michael Breza for early review and feedback. This work is supported by the UK EPSRC projects CHEDDAR (EP/X040518/1) and CHEDDAR Uplift (EP/Y037421/1), and an Amazon Research Award on Automated Reasoning.

REFERENCES

[1] Farooq Ahmad et al. "Formal modeling and analysis of security schemes of RPL protocol using colored Petri nets". In: *Plos one* 18.8 (2023), e0285700.

[2] Maram Albalwe, Blair Archibald, and Michele Sevegnani. "Modelling Real-time Systems with Bigraphs". In: *Proceedings. 15th International Workshop on Graph Computation Models,* Enschede, Netherlands, 8-11 July. 2024.

[3] Roger Alexander et al. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. Mar. 2012. DOI: 10.17487/RFC6550. URL: https://rfc-editor.org/rfc/rfc6550.txt.

[4] LoRa Alliance. "White Paper: A Technical Overview of LoRa and LoRaWAN". In: *The LoRa Alliance: San Ramon, CA, USA* (2015), pp. 7–11.

[5] Blair Archibald, Muffy Calder, and Michele Sevegnani. “Conditional Bigraphs”. In: *International Conference on Graph Transformation*. Springer. 2020, pp. 3–19.

[6] Blair Archibald, Muffy Calder, and Michele Sevegnani. “Probabilistic Bigraphs”. In: *Form. Asp. Comput.* 34.2 (2022). ISSN: 0934-5043. DOI: 10.1145/3545180. URL: https://doi.org/10.1145/3545180.

[7] Blair Archibald, Géza Kulcsár, and Michele Sevegnani. “A tale of two graph models: a case study in wireless sensor networks”. In: *Formal Aspects of Computing* 33.6 (2021), pp. 1249–1277.

[8] Adnan Aziz et al. “Verifying Continuous Time Markov Chains”. In: *Computer Aided Verification, 8th International Conference, CAV ’96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*. Ed. by Rajeev Alur and Thomas A. Henzinger. Vol. 1102. Lecture Notes in Computer Science. Springer, 1996, pp. 269–276. DOI: 10.1007/3-540-61474-5_75. URL: https://doi.org/10.1007/3-540-61474-5%5C_75.

[9] Hamid Bagheri et al. “A formal approach for detection of security flaws in the android permission system”. In: *Formal Aspects of Computing* 30 (2018), pp. 525–544.

[10] Steve Benford et al. “On Lions, Impala, and Bigraphs: Modelling Interactions in Physical/Virtual Spaces”. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 23.2 (2016), pp. 1–56.

[11] Bruno Blanchet et al. “ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial”. In: *Version from* (2018), pp. 05–16.

[12] Dominik Bork, Dimitris Karagiannis, and Benedikt Pittl. “A survey of modeling language specification techniques”. In: *Information Systems* 87 (2020), p. 101425.

[13] Rachida Boucebsi and Faiza Belala. “A Bigraphical Reactive Systems with Sharing for modeling Wireless Mesh Networks”. In: *journal of King Saud University-Computer and Information Sciences* (2018).

[14] Muffy Calder and Michele Sevegnani. “Modelling IEEE 802.11 CSMA/CA RTS/CTS with stochastic bigraphs with sharing”. In: *Formal Aspects of Computing* 26.3 (2014), pp. 537–561.

[15] Vincenzo Ciancia et al. “Model Checking Spatial Logics for Closure Spaces”. In: *Logical Methods in Computer Science* 12 (2017).

[16] Alessandro Cimatti et al. “NuSMV 2: An OpenSource Tool for Symbolic Model Checking”. In: *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*. Ed. by Ed Brinksma and Kim Guldstrand Larsen. Vol. 2404. Lecture Notes in Computer Science. Springer, 2002, pp. 359–364. DOI: 10.1007/3-540-45657-0_29. URL: https://doi.org/10.1007/3-540-45657-0%5C_29.

[17] Edmund M. Clarke and E. Allen Emerson. “Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic”. In: *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*. Ed. by Dexter Kozen. Vol. 131. Lecture Notes in Computer Science. Springer, 1981, pp. 52–71. DOI: 10.1007/BFb0025774. URL: https://doi.org/10.1007/BFb0025774.

[18] Manuel Clavel et al., eds. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. Vol. 4350. Lecture Notes in Computer Science. Springer, 2007. ISBN: 978-3-540-71940-3. DOI: 10.1007/978-3-540-71999-1. URL: https://doi.org/10.1007/978-3-540-71999-1.

[19] Christian Dehnert et al. “A Storm is Coming: A Modern Probabilistic Model Checker”. In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 592–600. DOI: 10.1007/978-3-319-63390-9_31. URL: https://doi.org/10.1007/978-3-319-63390-9%5C_31.

[20] Jannik Dreier et al. “Formally and practically verifying flow properties in industrial systems”. In: *Computers & Security* 86 (2019), pp. 453–470.

[21] Anwar Ghani et al. “Security and key management in IoT-based wireless sensor networks: An authentication protocol using symmetric key”. In: *International Journal of Communication Systems* 32.16 (2019), e4139.

[22] Rob van Glabbeek et al. “Modelling and verifying the AODV routing protocol”. In: *Distributed Computing* 29.4 (2016), pp. 279–315.

[23] Gerard J. Holzmann. “The Model Checker SPIN”. In: *IEEE Trans. Software Eng.* 23.5 (1997), pp. 279–295. DOI: 10.1109/32.588521. URL: https://doi.org/10.1109/32.588521.

[24] Daniel Jackson. “Alloy: A Lightweight Object Modelling Notation”. In: *ACM Trans. Softw. Eng. Methodol.* 11.2 (Apr. 2002), pp. 256–290. ISSN: 1049-331X. DOI: 10.1145/505145.505149. URL: https://doi.org/10.1145/505145.505149.

[25] Kurt Jensen. *Coloured petri nets*. Springer, 1987.

[26] Chris Karlof and David Wagner. “Secure routing in wireless sensor networks: Attacks and countermeasures”. In: *Ad hoc networks* 1.2-3 (2003), pp. 293–315.

[27] Hyung-Sin Kim et al. “Challenging the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL): A s Survey”. In: *IEEE Communications Surveys & Tutorials* 19.4 (2017), pp. 2502–2525.

[28] Mustafa Kocakulak and Ismail Butun. “An Overview of Wireless Sensor Networks Towards Internet of Things”. In: *2017 IEEE 7th annual computing and communication workshop and conference (CCWC)*. Ieee. 2017, pp. 1–6.

[29] Jean Krivine, Robin Milner, and Angelo Troina. “Stochastic bigraphs”. In: *Electronic Notes in Theoretical Computer Science* 218 (2008), pp. 73–96.

[30] Tomas Kulik et al. “A Survey of Practical Formal Methods for Security”. In: *Formal Aspects of Computing* 34.1 (2022), pp. 1–39.

[31] Marta Kwiatkowska, Gethin Norman, and David Parker. “PRISM 4.0: Verification of Probabilistic Real-time Systems”. In: *International conference on computer aided verification*. Springer. 2011, pp. 585–591.

[32] Sven Linker, Fabio Papacchini, and Michele Sevegnani. “Finite Models for a Spatial Logic with Discrete and Topological Path Operators”. In: *Leibniz International Proceedings in Informatics, LIPIcs* 202 (2021).

[33] Si Liu, Peter Csaba Ölveczky, and José Meseguer. “A framework for mobile ad hoc networks in real-time Maude”. In: *Rewriting Logic and Its Applications: 10th International Workshop, WRLA 2014, Held as a Satellite Event of ETAPS, Grenoble, France, April 5-6, 2014, Revised Selected Papers 10*. Springer. 2014, pp. 162–177.

[34] Simon Meier et al. “The TAMARIN prover for the symbolic analysis of security protocols”. In: *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*. Springer. 2013, pp. 696–701.

[35] Robin Milner. *The space and motion of communicating agents*. Cambridge University Press, 2009.

[36] Robin Milner, Joachim Parrow, and David Walker. “A Calculus of Mobile Processes, I”. In: *Inf. Comput.* 100.1 (1992), pp. 1–40. DOI: 10.1016/0890-5401(92)90008-4. URL: https://doi.org/10.1016/0890-5401(92)90008-4.

[37] Houssem E Mohamadi, Nadjia Kara, and Mohand Lagha. “Formal verification of RGR-SEC, a secured RGR routing for UAANETs using AVISPA, Scyther and Tamarin”. In: *Future Network Systems and Security: 4th International Conference, FNSS 2018, Paris, France, July 9–11, 2018, Proceedings 4*. Springer. 2018, pp. 3–16.

[38] George Oikonomou et al. “The Contiki-NG open source operating system for next generation IoT devices”. In: *SoftwareX* 18 (2022), p. 101089.

[39] Fredrik Osterlind et al. “Cross-Level Sensor Network Simulation with Cooja”. In: *Proceedings. 2006 31st IEEE conference on local computer networks*. IEEE. 2006, pp. 641–648.

[40] Alejandro Rodríguez, Lars Michael Kristensen, and Adrian Rutle. “Formal modelling and incremental verification of the MQTT IoT protocol”. In: *Transactions on Petri Nets and Other Models of Concurrency XIV* (2019), pp. 126–145.

[41] Kashif Saghar et al. “Formal modelling of a robust Wireless Sensor Network routing protocol”. In: *2010 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE. 2010, pp. 281–288.

[42] Hamza Sahli, Thomas Ledoux, and Éric Rutten. “Modeling Self-Adaptive Fog Systems Using Bigraphs”. In: *International Conference on Software Engineering and Formal Methods*. Springer. 2019, pp. 252–268.

[43] Michele Sevegnani, Blair Archibald, and Maram Albalwe. *Bigraphs Model and Simulation for RPL*. Zenodo, July 2024. DOI: 10.5281/zenodo.12783222. URL: https://doi.org/10.5281/zenodo.12783222.

[44] Michele Sevegnani and Muffy Calder. “BigraphER: rewriting and analysis engine for bigraphs”. In: *International Conference on Computer Aided Verification*. Springer. 2016, pp. 494–501.

[45] Michele Sevegnani and Muffy Calder. “Bigraphs with sharing”. In: *Theoretical Computer Science* 577 (2015), pp. 43–73.

[46] Michele Sevegnani et al. “Modelling and Verification of Large-Scale Sensor Network Infrastructures”. In: *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE. 2018, pp. 71–81.

[47] O. Tange. *GNU parallel 2018*. https://doi.org/10.5281/zenodo.1146014, 2018.

[48] Ta Vinh Thong and Levente Buttyán. “On automating the verification of secure ad-hoc network routing protocols”. In: *Telecommunication Systems* 52 (2013), pp. 2611–2635.

[49] Pascal Thubert. *Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)*. RFC 6552. Mar. 2012. DOI: 10.17487/RFC6552. URL: https://www.rfc-editor.org/info/rfc6552.

[50] Christos Tsigkanos et al. “On the Interplay Between Cyber and Physical Spaces for Adaptive Security”. In: *IEEE Trans. Dependable Secur. Comput.* 15.3 (2018), pp. 466–480. DOI: 10.1109/TDSC.2016.2599880. URL: https://doi.org/10.1109/TDSC.2016.2599880.

[51] Ishaq Unwala, Zafar Taqvi, and Jiang Lu. “Thread: An IoT Protocol”. In: *2018 IEEE Green Technologies Conference (GreenTech)*. IEEE. 2018, pp. 161–167.

[52] Linus Wallgren, Shahid Raza, and Thiemo Voigt. “Routing Attacks and Countermeasures in the RPL-Based Internet of Things”. In: *International journal of Distributed Sensor Networks* 9.8 (2013).

[53] MP Webster et al. “Formal Verification of Synchronisation, Gossip and Environmental Effects for Critical IoT Systems”. In: *International Workshop on Automated Verification of Critical Systems (AVoCS)* (2018).

[54] Geoffrey Werner-Allen et al. “Deploying a Wireless Sensor Network on an Active Volcano”. In: *IEEE internet computing* 10.2 (2006), pp. 18–25.

[55] Tsu-Yang Wu et al. “A Provably Secure Three-Factor Authentication Protocol for Wireless Sensor Networks”. In: *Wireless Communications and Mobile Computing* 2021 (2021), pp. 1–15.

[56] Yan Xiong et al. “SmartVerif: Push the Limit of Automation Capability of Verifying Security Protocols by Dynamic Strategies”. In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 253–270.

Appendix A ALGEBRAIC DEFINITION OF THE RULES IN SECTION 4

Instantiation maps are indicated by η.

$$
\begin{aligned}
\texttt{generateDIO} &\stackrel{\text{def}}{=} \mathsf{Node}_n.(\mathsf{GenDIO} \mid \mathsf{Rank}.id_0 \mid id_1) \\
&\quad \longrightarrow\!\!\blacktriangleright \mathsf{Node}_n.(\mathsf{Send.DIO}_n.\mathsf{Rank}.id_0 \mid \mathsf{Rank}.id_1 \mid id_2) \\
&\quad \text{with } \eta = [0 \to 0, 1 \to 0, 2 \to 1] \\
\texttt{sendDIO} &\stackrel{\text{def}}{=} \mathsf{Node}_{n1}.id_0 \mid \mathsf{Node}_{n2}.(\mathsf{Send.DIO}_n.id_1 \mid id_2) \\
&\quad \parallel /x\,(\mathsf{PNode}_{n1}.(\mathsf{PLink}_x.id_3 \mid id_4) \mid \mathsf{PNode}_{n2}.(\mathsf{PLink}_x.\mathsf{Idle} \mid id_5)) \\
&\quad \longrightarrow\!\!\blacktriangleright \mathsf{Node}_{n1}.(\mathsf{Receive.DIO}_n.id_0 \mid id_1) \mid \mathsf{Node}_{n2}.(\mathsf{Send.DIO}_n.id_2 \mid id_3) \\
&\quad \parallel /x\,(\mathsf{PNode}_{n1}.(\mathsf{PLink}_x.id_4 \mid id_5) \mid \mathsf{PNode}_{n2}.(\mathsf{PLink}_x.\mathsf{In\text{-}use} \mid id_6)) \\
&\quad \text{with } \eta = [0 \to 1, 1 \to 0, 2 \to 1, 3 \to 2, 4 \to 3, 5 \to 4, 6 \to 5] \\
\texttt{sendDIO\_Done} &\stackrel{\text{def}}{=} \mathsf{Node}_n.(\mathsf{Send.DIO}_n.id_0 \mid id_1) \parallel \mathsf{PNode}_n.id_2 \\
&\quad \longrightarrow\!\!\blacktriangleright \mathsf{Node}_n.id_0 \parallel \mathsf{PNode}_n.(\mathsf{ClearMulticast} \mid id_1) \\
&\quad \text{with } \eta = [0 \to 1, 1 \to 2] \\
&\quad \text{if } \langle -, \mathsf{Idle}, \downarrow \rangle \\
\texttt{clearMulticast} &\stackrel{\text{def}}{=} \mathsf{PNode}_n.(\mathsf{ClearMulticast} \mid \mathsf{PLink}_x.\mathsf{In\text{-}use} \mid id) \\
&\quad \longrightarrow\!\!\blacktriangleright \mathsf{PNode}_n.(\mathsf{ClearMulticast} \mid \mathsf{PLink}_x.\mathsf{Idle} \mid id) \\
\texttt{clearMulticastEnd} &\stackrel{\text{def}}{=} \mathsf{PNode}_n.(\mathsf{ClearMulticast} \mid id) \longrightarrow\!\!\blacktriangleright \mathsf{PNode}_n.id \\
&\quad \text{if } \langle -, \mathsf{In\text{-}use}, \downarrow \rangle \\
\texttt{handleDIONotJoined} &\stackrel{\text{def}}{=} \mathsf{Node}_n.(\mathsf{Receive.DIO}_p.\mathsf{Rank}.id \mid id) \\
&\quad \longrightarrow\!\!\blacktriangleright \mathsf{Node}_n.(\mathsf{IncRank.Rank}.id \mid \mathsf{PrefPrnt}_p \mid \mathsf{Send.DAO}_{n,p} \mid id) \\
&\quad \text{if } \langle -, \mathsf{Rank}, \downarrow \rangle \\
\texttt{handleDIOJoined} &\stackrel{\text{def}}{=} \mathsf{Node}_n.(\mathsf{Receive.DIO}_p.\mathit{Rank}.id_0 \mid \mathsf{Rank}.id_1 \mid id_2) \\
&\quad \longrightarrow\!\!\blacktriangleright \mathsf{Node}_n.(\mathsf{LT}.(\mathsf{L.Rank}.id_0 \mid \mathsf{R.IncRank.Rank}.id_1) \mid \mathsf{Receive.DIO}_p.\mathsf{Rank}.id_2 \mid \mathsf{Rank}.id_3 \mid id_4) \\
&\quad \text{with } \eta = [0 \to 1, 1 \to 0, 2 \to 0, 3 \to 1, 4 \to 2] \\
&\quad \text{if } \langle -, \mathsf{LT}, \downarrow \rangle \\
\texttt{handleDIOLT} &\stackrel{\text{def}}{=} \mathsf{Node}_n.(\mathsf{LT.True} \mid \mathsf{Receive.DIO}_p.\mathsf{Rank}.id_0 \mid id_1) \longrightarrow\!\!\blacktriangleright \mathsf{Node}_n.id_0 \mid p \\
&\quad \text{with } \eta = [0 \to 1] \\
\texttt{handleDIOGT} &\stackrel{\text{def}}{=} \mathsf{Node}_n.(\mathsf{PrefPrnt}_x \mid \mathsf{LT.False} \mid \mathsf{Rank}.id_0 \mid \mathsf{Receive.DIO}_p.\mathsf{Rank}.id_1 \mid id_2) \\
&\quad \longrightarrow\!\!\blacktriangleright \mathsf{Node}_n.(\mathsf{Send.DAO}_{n,p} \mid \mathsf{PrefPrnt}_p \mid \mathsf{IncRank.Rank}.id_0 \mid id_1) \mid x \\
&\quad \text{with } \eta = [0 \to 1, 1 \to 2]
\end{aligned}
$$

$$\texttt{ranksComparison(n,m)} \stackrel{\text{def}}{=} \mathsf{LT}.(\mathsf{L.Rank.Val(n)} \mid \mathsf{R.Rank.Val(m)}) \longrightarrow \mathsf{LT}.(\mathsf{L.Rank.Val(n\text{-}1)} \mid \mathsf{R.Rank.Val(m\text{-}1)})$$

$$\texttt{incRank(m)} \stackrel{\text{def}}{=} \mathsf{IncRank.Rank.Val(m)} \longrightarrow \mathsf{Rank.Val(m{+}1)}$$

$$\texttt{lt\_l\_0} \stackrel{\text{def}}{=} \mathsf{LT}.(\mathsf{L.Rank.Val(0)} \mid id_0) \longrightarrow \mathsf{LT.True} \quad \text{with } \eta = [\,]$$

$$\texttt{lt\_r\_0} \stackrel{\text{def}}{=} \mathsf{LT}.(\mathsf{R.Rank.Val(0)} \mid id_0) \longrightarrow \mathsf{LT.False} \quad \text{with } \eta = [\,]$$

$$\texttt{sendDAO} \stackrel{\text{def}}{=} (\mathsf{Node}_{n1}.(\mathsf{Send.DAO}_{n1,n2} \mid id) \mid \mathsf{Node}_{n2}.id) \longrightarrow (\mathsf{Node}_{n1}.(\mathsf{GenDIO} \mid \mathsf{Joined} \mid id) \mid \mathsf{Node}_{n2}.(\mathsf{Receive.DAO}_{n1,n2} \mid id))$$

$$\texttt{sendDAOUpParent} \stackrel{\text{def}}{=} \mathsf{Node}_{n2}.(\mathsf{PrefPrnt}_{n1} \mid \mathsf{Receive.DAO}_{n2,f} \mid id) \mid \mathsf{Node}_{n1}.id \longrightarrow \mathsf{Node}_{n2}.(\mathsf{PrefPrnt}_{n1} \mid id) \mid \mathsf{Node}_{n1}.(\mathsf{Receive.DAO}_{n1,n2} \mid f \mid id)$$

$$\texttt{clearDAO} \stackrel{\text{def}}{=} \mathsf{Node}_{n1}.(\mathsf{Receive.DAO}_{n1,n2} \mid id) \longrightarrow \mathsf{Node}_{n1}.(n2 \mid id)$$

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009