Timed Bigraphs for Formal Verification of Sensor Network Routing Protocols

Maram Mohammed Shaher Albalwe

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy

School of Computing Science
College of Science and Engineering
University of Glasgow

February 2025

Abstract

Given that more end-user applications depend on the Internet of things (IoT) technology, which relies heavily on wireless sensor networks (WSNs), it is essential that the routing protocols underpinning these applications are reliable. Using formal methods for reasoning on protocol specifications is an established technique but, due to their perceived difficulty and mathematical nature, they receive limited use in practice. This thesis proposes 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 the use of formal methods in protocol design. Bigraphical reactive systems (BRSs) are a graph-rewriting formalism describing systems evolving in two dimensions: spatially, *e.g.* a person in a room, and non-spatially, *e.g.* mobile phones communicating regardless of location. To show bigraphs in action, this thesis models part of the routing protocol for low-power and lossy networks (RPL), popular in wireless sensor networks. The model is implemented using the BigraphER toolkit and verified with the PRISM model checker.

Simulation, on the other hand, is a common approach in the field of protocol analysis and validation. However, it does not extensively verify protocols in the same way as formal methods do. This thesis experimentally compares the two approaches, the results of which show that analysing the bigraph model often finds more valid routes than simulation while providing comparable performance. The bigraphs model is open to extension with less implementation effort than simulation, which is shown by adding more features to the initial model. Bigraphs seem to be a promising approach for protocol design; this is the first step in promoting their use.

Despite the use of bigraphs in domains that include communication protocols, agent programming, biology, and security, there is no support for real-time systems. Therefore, this thesis extends BRSs to support real-time systems by using a modelling approach that employs multiple perspectives to represent digital clocks. It uses Action BRSs, a recent extension of BRSs, where the resulting transition system is a Markov decision process (MDP). This allows a natural representation of the choices in each system state: to either allow time to pass or perform a specific action. The effectiveness of this approach is demonstrated using examples, including extending the RPL initial model with timed aspects using the BigraphER toolkit.

Contents

List of Tables

List of Figures

Acknowledgements

I am deeply grateful for the constant support and guidance of my supervisor, Dr. Michele Sevegnani. His insightful suggestions, constructive feedback and encouragement have been invaluable throughout my PhD journey. I would like to express my special thanks to Dr. Blair Archibald, for his invaluable support, perceptive feedback and continuous motivation, which are greatly appreciated.

I would like to thank the University of Tabuk for providing me with a scholarship and financial support during my studies. Furthermore, I would like to acknowledge the Saudi Arabian Cultural Bureau in London for their continuous support during my academic journey.

I owe an immeasurable debt of gratitude to my beloved mother, Amaal, for her support, endless prayers and encouragement, which support me through every challenge. Her faith in me and boundless love have been the cornerstone of my life. I want to express my deepest and most heartfelt gratitude to my husband, Yasser, for his unwavering love, support and understanding that have been a constant source of strength. Yasser, I could not have made it this far without you by my side. I give special thanks to my lovely children, Yasmeen and Farah, whose cheerfulness and smiles have brought light to the most challenging days, reminding me of the beauty and joy in life.

Heartfelt thanks go to my amazing siblings for their support, encouragement and keeping me present in their moments. I would like to extend my thanks to my mother-in-law for her prayers and kind motivated words. To my sincere friends back in Saudi Arabia, thank you for your support and for always including me in your gatherings, even from afar.

Finally, I sincerely thank my examiners, Prof. Pierre-Loïc Garoche and Dr. Emma Li, for their time and insightful feedback on my thesis.

I dedicate this work to my late father, Mohammed Shaher, whose encouragement and belief in the power of education inspired me to pursue my postgraduate studies. I started this journey with the hope of making you happy and proud, and now words do not suffice to describe how I feel about reaching this moment without you. This accomplishment is as much yours as it is mine, and I am forever grateful for the love, wisdom and support you gave me. Yara, my little angel, you are in our thoughts every single day. To the loving memory of my grandmother, whose words and prayers continue to surround, even in her absence.

Chapter 1

Introduction

1.1 Overview

The Internet of things (IoT) refers to a network that consists of physical items equipped with sensors, software and diverse technologies that allow them to connect, gather and share data via the Internet. Wireless sensor networks (WSNs) [71] are a key enabling technology for IoT through its sensor nodes due to their ability to sense data in the surroundings, process it and then transmit it to the intended destination. The sensor nodes are typically low-power and low-cost devices that use wireless communication. That is, in an IoT system, WSNs are used to collect data from the physical environment and transmit it to a centralised system such as a cloud service for processing and storage. Due to the extensive applicability of WSNs and the increasing need for monitoring, tracking and automation in different industries *e.g.*, [42, 14, 109, 65], WSNs demonstrate rapid growth. According to the International Data Corporation (IDC), 55.7 billion IoT devices will be connected to the Internet by 2025 [62]. In addition, Markets and Markets Reports state that the smart sensors market is expected to grow to USD 136.3 billion by 2029 [91].

Routing is a critical component of WSNs in IoT, which is the process of determining the path through which data is transmitted from one sensor node to another or to a central base station. Sensor nodes are often deployed in a distributed area or in a harsh environment, *e.g.*, a volcanic area [152] that requires efficient routing techniques to ensure reliable data delivery. Additionally, sensor nodes are usually powered by a small battery, hence routing algorithms should consider energy efficiency. Due to the diversity of WSN devices, routing protocols should be robust enough to withstand or prevent attacks. Therefore, routing algorithms are vital for assessing the efficiency and resilience of WSNs.

Traditionally, protocol designers validate routing protocols through simulation. The simulation approach creates an abstract model to analyse the behaviour of protocols. It provides a clear understanding and analysis of protocols by performing a large number of tests under different scenarios. Although simulation helps to understand and analyse the proposed protocols, it does not provide full coverage of all the potential outcomes [46, 47]. Consequently, it does not guarantee the absence of bugs. In critical systems, such as WSNs, verifying every possible route is essential. Formal verification approaches, in contrast, perform an exhaustive check for all the possible combinations, providing full coverage of the state space, which leads to the discovery of hidden bugs and errors.

Formal methods are mathematical techniques that are used to model and reason about systems. They check completeness, correctness and inconsistency. By formally analysing systems under different scenarios, one can detect vulnerabilities, attacks and errors in the early stages of the design. Formal methods provide abstraction for a system, *i.e.*, enable a high-level representation, which assists a rigorous analysis. Therefore, they help in understanding, analysing and predicting the behaviour of the system. Formal methods allow for overall quality improvement by providing efficient test scenarios, which also guarantee that a system satisfies its requirements. Due to the clear and constant description they provide for systems, formal methods ease the maintainability and reusability.

1.2 Motivation

Despite the advantages of formal methods, they receive less usage compared with simulation. One reason behind this could be that many of the formal modelling formalisms and tools heavily rely on mathematical foundations, which are often seen as difficult to use and the uptake is slow. The trade-off between the benefits of formal methods in guaranteeing behaviour and their usability is a pressing question in protocol design; the Internet Engineering Task Force (IETF) recently proposed their *Usable Formal Methods Research Group*[1]. Given the importance of WSN routing protocols, the limitation of the simulation approach, and the difficulties seen in formal methods, this thesis introduces a *diagrammatic* formal method, *i.e.*, bigraphical reactive systems (BRSs), to show the practical application of Milner's bigraphs as a modelling technique for protocol design that allows for usability without sacrificing rigour.

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, such as π-calculus [98]) 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 [10].

This thesis presents such an approach by modelling and analysing RPL [5] – a routing protocol for low-power and lossy networks, one of the most well-known and widely used

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

protocols. Given the shared characteristics of RPL with other protocols, a similar model is expected for a much wider range of network protocols, as can be seen by similar models for the 802.11 MAC protocol [24] and other WSN topology protocols [10]. The model is verified using model checking, *i.e.*, PRISM. In addition, an experiment comparison is performed between the RPL bigraph models versus the well-studied RPL implementation, *i.e.*, RPL-Lite[2].

The standard theory of bigraphs has been extended, *e.g.*, stochastic bigraphs [75] assign rates to reaction rules, bigraphs with sharing [128] allow for intersecting locations, directed bigraphs [49] associate directions to links, conditional bigraphs [7] add conditions to rules, and probabilistic and action bigraphs [9] support non-determinism and probabilistic behaviour. One extension that has not been explored is real-time bigraphs that can model systems exhibiting non-deterministic behaviour that is controlled by time constraints. While the non-deterministic aspects of real-time systems could be modelled by action bigraphical reactive systems (ABRSs) [9], in which the underlying Markov decision process (MDP) semantics allows for a transition choice at each state, there is currently no notion of *clocks constraints* in the theory, *e.g.*, specifying that after 3 time units have passed, a given action *must* occur. Given routing protocols (usually timing aspects), this thesis extends bigraphs with a modelling strategy to encode timed systems within ABRSs through the well-known MDP-based digital clock approximation.

1.3 Research Questions

This thesis contributes to the current research efforts that aim to improve the design of IoT protocols. In addition, it circumvents the limitations that formal models may have, *i.e.*, when modelling timed aspects. This thesis answers the following questions.

- How do formal methods differ from simulation, the popular validation approach, in the protocol analysis field?
- Can protocol designers improve their approach of analysing newly proposed protocols by using formal methods approaches?
- Does the introduction of bigraphs contribute to the protocol design and verifying?
- Are the standard bigraphs a good choice in modelling WSNs routing protocols?
- To what extent can bigraphs be extended to model the characteristics of WSN routing protocols?
- Do bigraphs rewriting tools provide efficient practical models for WSN routing protocols?

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

1.4 Thesis Statement

This thesis proposes bigraphs for modelling and reasoning in relation to wireless sensor networks (WSNs) routing protocols. A key goal of routing is to build an efficient route of communication between the gateway and each sensor to enable reliable data transfer [69]. This thesis examines an approach that considers the routing protocol for low-power and lossy networks (RPL) [5] as an example, while the belief is that the same approach can be applied to other WSNs protocols. In addition, this thesis investigates the feasibility of extending bigraphs with clocks to model timing aspects that routing protocols often contain.

Hypothesis. This research investigates the use of bigraphs to formally model and verify WSNs protocols. Bigraphs are an expressive graph-rewriting formalism based on relating entities both spatially, through nesting, and with non-local connectivity that can mirror sensor networks that have a strong spatial component, *e.g.*, the radio range determining the physical neighbour set, but also virtual connectivity, *e.g.*, determining multi-hop paths from a sensor to base states. A key benefit of using 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, *e.g.*, those presented in *e.g.*, [105, 119]. Moreover, bigraphs have been extended to several variants. Hence, this thesis investigates extending bigraphs with clocks to model real-time aspects of the routing protocols.

1.5 Thesis Contributions

The main contributions of this thesis are as follows:

- Novel bigraphical models for RPL key component, *i.e.*, the destination oriented directed acyclic graph (DODAG). RPL is one of the most used WSNs routing protocols, it has been extensively analysed using simulators. In this thesis, we investigate using bigraphs to formally model and analysis RPL. This thesis utilises model checking that shows the RPL model maintains useful properties, *e.g.*, routes are valid, and they are cycle-free. An executable specification of the model is provided in the BigraphER [127] framework.

- Comparison of the RPL bigraphical model with the existing simulation, a popular analysing approach for RPL. This thesis experimentally compares the bigraph model with the RPL-Lite[3] implementation in the Contiki-NG operating system [106]. The simulation is performed using the Cooja network simulator. This comparison is based on the number of routes found and the running time. The results reveals 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 has comparable performance.

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

- The modelling of real-time systems with bigraphs. A modelling strategy is proposed to express clocks within action bigraphs to enable time-aspect modelling in bigraphs. This thesis defines a set of reaction rules to model clock constraints in real-time systems through a digital clocks approximation. It illustrates how to use the proposed approach in practice by providing examples including a model of some timed aspects of a real-world network protocol, *i.e.*, RPL. The implementation of this approach is also provided in the BigraphER toolkit to show that this is not just a theoretical contribution, but a practical one.

1.6 Thesis Publications

The contents of Chapters 5, 6 and 7 are presented in the following peer-reviewed publications:

- **Maram Albalwe**, Blair Archibald and Michele Sevegnani. "Modelling and Analysing Routing Protocols Diagrammatically with Bigraphs". In: Formal Aspects of Computing 36.3 (2024), pp. 1–25, DOI. 10.1145/3685934.

- **Maram Albalwe**, Blair Archibald and Michele Sevegnani. "Modelling Real-Time Systems with Bigraphs". In: Proceedings of the 15th International Workshop on Graph Computation Models (2024).

Two more publications (based on the research in this thesis) are expected to be produced. These are described as follows:

- The second paper (above) presents preliminary results of the bigraphs extension with the clocks. The paper briefly shows that the RPL time aspects can be modelled with the proposed technique. In addition, the first paper presents the initial model for the RPL protocol. Hence, the plan is to update the RPL model with the extensions presented in Chapters 5 and 7, including the timed model.

- The RPL model is extendable and its ability to include security attacks is illustrated in Chapter 5. Therefore, the plan is to formally analyse and reason about security attacks that threaten RPL to investigate bigraphs ability to detect and prevent vulnerabilities in WSNs.

1.7 Thesis Structure

This thesis comprises 8 chapters and 2 appendices. Chapter 2 outlines an overview of the theoretical concepts of wireless sensor networks along with their routing concept. It also provides a description of RPL and its main features. In addition, the main validation approaches that are used in WSNs are covered. Next, Chapter 3 presents a description of formal verification approaches; a state-of-the-art system is presented, through a comparison between formal verification and simulation approaches, alongside various existing works on the application of formal methods on WSNs. Chapter 4 provides preliminary concepts that are used in the later chapters. It delves into the essential background for bigraphical reactive systems (BRSs), the proposed formal method applied in this thesis. The RPL bigraphical models are extensively demonstrated in Chapter 5. Then, Chapter 6 verifies the RPL model and experimentally compares the bigraphical model and the simulation. Chapter 7 illustrates how this thesis extends the standard bigraphs by introducing clocks to enable bigraphs to model real-time systems. In addition, the bigraphical RPL model is extended to include some of the RPL timing aspects. Finally, Chapter 8 discusses the findings and presents the conclusions and recommendations for future work. The implementations of the bigraph models are provided in the appendices.

Chapter 2

Wireless Sensor Networks

This chapter presents an overview of wireless sensor networks (WSNs), highlighting their key features and applications (Section 2.1). Then, WSN routing concepts and algorithms are provided in Section 2.2, while the routing protocol for low-power and lossy networks (RPL) is elaborated upon in Section 2.3 since it is used as a case study in this thesis. Section 2.4 demonstrates the WSNs analysing approaches: simulation, which is widely used as a reliable analysis approach for WSNs (Section 2.4.1), and a brief introduction to the formal verification approach (Section 2.4.2). Due to the focus of this thesis on the formal verification of WSN routing protocols, the next chapter discusses this approach in more detail. The chapter is summarised in Section 2.5.

2.1 Wireless Sensor Networks

IoT consists of a large number of devices from different manufacturers that have different capabilities. However, they should operate together without compromising performance. The devices used in the IoT environment are often powered by battery and need to work continuously for long periods. IoT systems should also operate in dynamic and unpredictable environments. Consequently, the IoT suffers from a variety of types of attacks and vulnerabilities that threaten its security and reliability. In addition, the highlighted characteristics contribute to the complexity of the IoT, which subsequently complicates its development, testing and maintenance [74]. Since the evolution of IoT can be seen as a result of the rapid development of communication technologies, the development of these technologies should consider the features and limitations of the IoT. One such technology is wireless sensor networks (WSNs), which enable IoT through interconnected devices, such as sensors and actuators. That is, digitising physical-world data requires sensor nodes (nodes for short) that are able to sense, collect, process and store data in a digital form.

The nodes are small embedded systems used to monitor and control physical and environmental variables. Some nodes provide more functionalities, such as the ability to extract only relevant data, which results in reducing post-processing latency [135]. The nodes also usually have different capabilities in terms of monitoring, communication and their ability to access

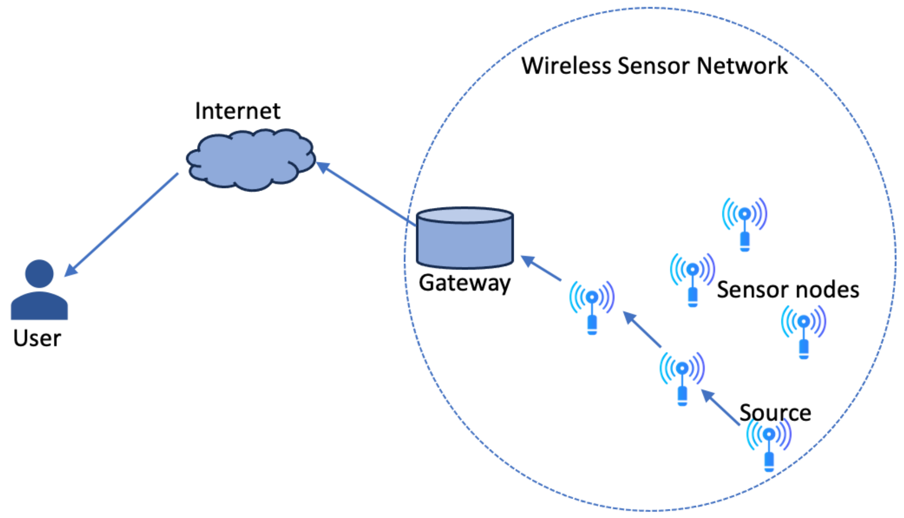


Figure 2.1: A common wireless sensor network configuration.

more supporting technologies. For example, multiple sensor types – including the DHT11 sensor, soil temperature sensor and soil moisture sensor – are installed in a greenhouse at an agriculture research centre to monitor and measure temperature and humidity [109]. The nodes send the collected data to the Thing Speak cloud service for storage and display purposes. WSNs have been used in a wide range of applications. In health-related scenarios, a sensing health with intelligence, modularity, mobility and experimental reusability (SHIMMER) mote, which is designed for WSNs health and biomedical applications, is placed inside a wrist cuff that is connected to a pressure sensor in the Heart@Home system to calculate blood pressure and heart rate [14]. For tracking purposes, Max, for example, is a system that a human uses to search and locate their physical objects by providing the location information reference of nearby landmarks, *e.g.*, on the dining table [158]. ZebraNet is another tracking application that is a mobile wireless sensor network developed to track animal migrations. It sends the positional readings obtained using the GPS to the base station [65]. The applications of WSNs have also been deployed in underwater environments for monitoring, military, navigation and sports purposes [42]. Other applications require a more challenging deployment, such as sensors mounted in a harsh environment, *i.e.*, nodes within a volcanic environment [152].

Many real-world applications deploy nodes in a large area that aim to provide an effective sense of the environment. Although 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 (Figure 2.1). Since these nodes usually reduce their transmission power, which consequently decreases their communication range to conserve battery life, multi-hop communication between nodes is required. Therefore, special routing protocols are needed for WSNs.

2.2 Routing in Wireless Sensor Networks

A key goal of WSN routing is to build an efficient route of communication between the gateway and each node to allow for reliable data transfer. When all the nodes have large transmission ranges, a single-hop communication is formed between any node and the gateway, which forms a star topology. However, low-cost nodes often have low-powered transmitters that are out-of-range of a gateway node, meaning multi-hop routing formation, which results in different types of topology. In this multi-hop routing, data is sent from nodes to other nodes close to the gateway [149].

The multi-hop communication approach is a necessity for most WSN applications where the distance, which is the number of hops, between the nodes and the gateway is reduced. However, this results in another issue, *i.e.*, the selection of an optimal route from a node to the gateway. Different multi-hop strategies exist to find the optimal route that decreases energy consumption [40]. These techniques are classified under multi-hop clustering techniques and load-balance tree techniques. Clustering techniques operate according to the divide-and-conquer principle; therefore, they are considered as the best technique for heterogeneous WSNs. In other words, exploiting the diversity of node capabilities results in balancing the load between nodes, *e.g.*, low-power nodes transmit over limited distances, while high-capacity nodes are responsible for extended multi-hop communications between clusters. The load-balanced tree, on the other hand, finds the path that is energy-balanced and energy-efficient through the multipath technique or the single-path technique [40].

2.2.1 WSNs Routing Protocols

The routing protocols specifically developed for WSNs differ from each other depending on the type of application. Some applications require data to be sent immediately when detected. In such a system, the consumption of energy and computation resources for nodes is decreased since the communication occurs only if significant data is detected. As a result of a random trigger of data, the workloads will be unbalanced and, thus, some nodes will expire before others, leading to isolated areas. On the other hand, time-driven protocols have been developed for applications in which the sensed data is periodically sent; thus, the nodes sleep between transmissions and energy is conserved [162]. Since WSNs nodes are relatively small and operate on a battery that is usually difficult to recharge, routing protocols are supposed to operate with a minimum use of energy to extend the lifetime of the networks.

The design of a routing protocol that is energy efficient is an important factor [20]. To achieve this efficiency, it is important to consider the spatial distribution and heterogeneity of the nodes when designing routing protocols for WSNs. When all the nodes have similar functionality, communication and energy consumption, the WSN then requires flat routing algorithms where equal treatment is given to all the nodes. In contrast, a hierarchical routing algorithm is where the nodes are divided into multiple clusters, in which the higher power level node is selected to be a cluster head for that cluster. Communication is then established between the nodes and the cluster head in the related cluster. Each cluster head communicates with the gateway, leading to faster and more efficient energy consumption. A cluster head may send data directly to the gateway or pass it on to the next cluster head until it is delivered to the gateway [40, 25].

The way in which a node selects the next hop differs from one protocol to another. Nodes in broadcast-based protocols broadcast packets to every neighbour node that re-broadcasts the packet further. Geographic information is used by nodes in location-based protocols to select the next hop. This way of selecting the next hop reduces the number of packet transmissions. The sensed data sometimes contain the routing information, which is used by the node to select the next hop in content-based protocols. Probabilistic routing protocols randomly select the next hop from the available nodes since they assume that all nodes are randomly distributed [162]. Distance vector routing protocols use a notion of *distance* (or distance proxy such as latency) to determine the best path for a packet in terms of the number of hops.

The route in WSNs may be requested by a node that needs a service from the gateway or to send data to the supposed destination. However, the gateway can also initiate the route establishment process when it needs to send a packet to a node. Routing protocols in WSNs have three route-finding approaches: proactive, reactive and hybrid. In proactive protocols, the route is identified before it is required by advertising the routing table in the memory of each node. In reactive routing protocols, the route is only computed when needed, while hybrid protocols select the most efficient routing technique for packet delivery that accelerates transmission and reduces processing overhead [162].

The nature of applications and their requirements should be considered when designing and selecting routing protocols. For example, when an application is required to be deployed in a harsh environment, in which the nodes cannot be replaced or recharged, the selection of routing protocols must consider minimising energy consumption. However, in other applications where node failures should not affect the sensed data delivery, the routing protocol should involve failure recovery mechanisms [116].

Ultimately, routing protocols should have an efficient routing algorithm that, *e.g.*, conserves node energy, which increases the reliability of the whole network. In addition, routing protocols should also be scalable for different network sizes ranging from tens to thousands of nodes. Moreover, they should be able to monitor real-time topology and allow routing to follow any changes in node organisation. Routing protocols should also be designed to be secure, free of vulnerability and resistant to attacks [25].

2.3 Routing Protocol for Low-Power and Lossy Networks (RPL)

This thesis considers the routing protocol for low-power and lossy networks (RPL) to showcase how routing protocols are formally verified. RPL is a widely used WSN routing protocol, defined by IETF RFC 6550 [5] and implemented for many WSN toolkits, most notably Contiki-NG. It is a distance-vector proactive routing protocol designed especially for low-power and lossy networks (LLNs). RPL is effective in managing undefined and unstable topologies with some timing aspects.

RPL is a popular protocol that has been exhaustively analysed. Numerous articles and reviews discuss different RPL-related topics, *e.g.*, implementations, security, enhancements, etc. [35, 79, 66]. It has been utilised in a range of applications, such as healthcare, smart transportation and military [69]. RPL is designed to satisfy the specification of low-power and lossy networks. It is considered to be an efficient and appropriate routing protocol for such networks [4]. LLN characteristics are described in RFC6550 as follows:

1. LLN consists of routers and nodes that both have restricted processing power, energy and memory.

2. The nodes are unstable and feature high packet loss rates with low data rates.

3. The number of routers ranges from a few dozen to thousands.

4. LLNs support various types of traffic patterns. Namely, point-to-point, multipoint-to-point, or point-to-multipoint.

RPL has self-healing and repair approaches, *e.g.*, local and global repairs and loop detection mechanism. RPL has built-in three security modes: unsecured, pre-installed and authenticated modes. As the first mode name indicates, the messages are exchanged without a security mechanism. However, in the pre-installed mode, messages are secured with keys, thus data confidentiality and integrity are maintained. The last mode is similar to the pre-installed mode, where nodes have preinstalled keys, but it is different in that the key can be used to join as a leaf node.

However, RPL is vulnerable to numerous types of attacks [94]. One reason is that RPL nodes are usually constrained by power, memory, etc., so they can be easily compromised [90]. RPL is vulnerable to general WSN attacks or attacks that targeting it specifically [147]. In addition, threats could be from both outside and inside the network. Outsiders present on the Internet may sniff or spoof data from the network into the nodes. Insiders originate from the network nodes since they may have some faults, misconfiguration or some physical tampering of the IoT devices [90].

2.3.1 RPL Routes Construction

Since LLNs do not have a predefined topology, RPL must find links and construct the topology. Given a physical network topology, *e.g.*, sensor positions and radio ranges and a predefined *root* node, usually a gateway between 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. If there exists more than one root, a common backbone that enables the roots to federate should be created. As shown in Figure 2.2, RPL defines four control messages:

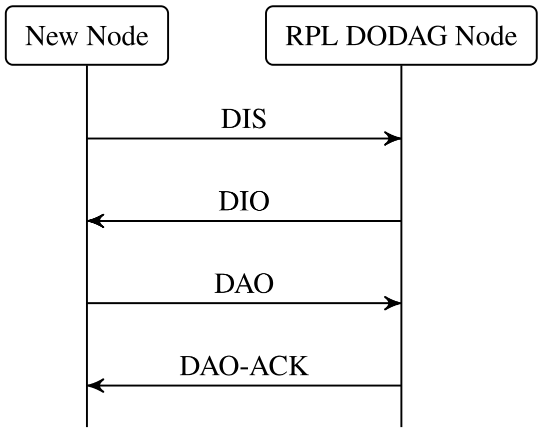


Figure 2.2: RPL control messages.

- DIS (DODAG information solicitation). This 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). This transmits information, *e.g.*, rank, to other nodes regardless of whether they have already joined the DODAG.
- DAO (destination advertisement object). This 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). This is sent by the root to a joining node that, upon receiving it, becomes reachable and can multicast DIOs to other nodes.

RPL DODAGs Construction

RPL allows different directions of routing: (a) downwards routing from the root to any node, (b) upwards routing from any node to a root and (c) any-to-any routing between arbitrary pairs of nodes. We illustrate the protocol by informally describing the steps required to construct its key element, *i.e.*, a DODAG. We will use the abbreviation DAG (instead of DODAG) for the rest of this thesis.

Figure 2.3 demonstrates the DAG construction process. For each sub-figure, an RPL node (in blue) multicasts a DIO (then turns red) and nodes in its range join (in light red). Note that grey edges indicate *physical links* between nodes, red dashed arrows signify *links used by the RPL node in multicasting DIOs*, and black solid arrows denote *built DAG routes*. The process is as follows:

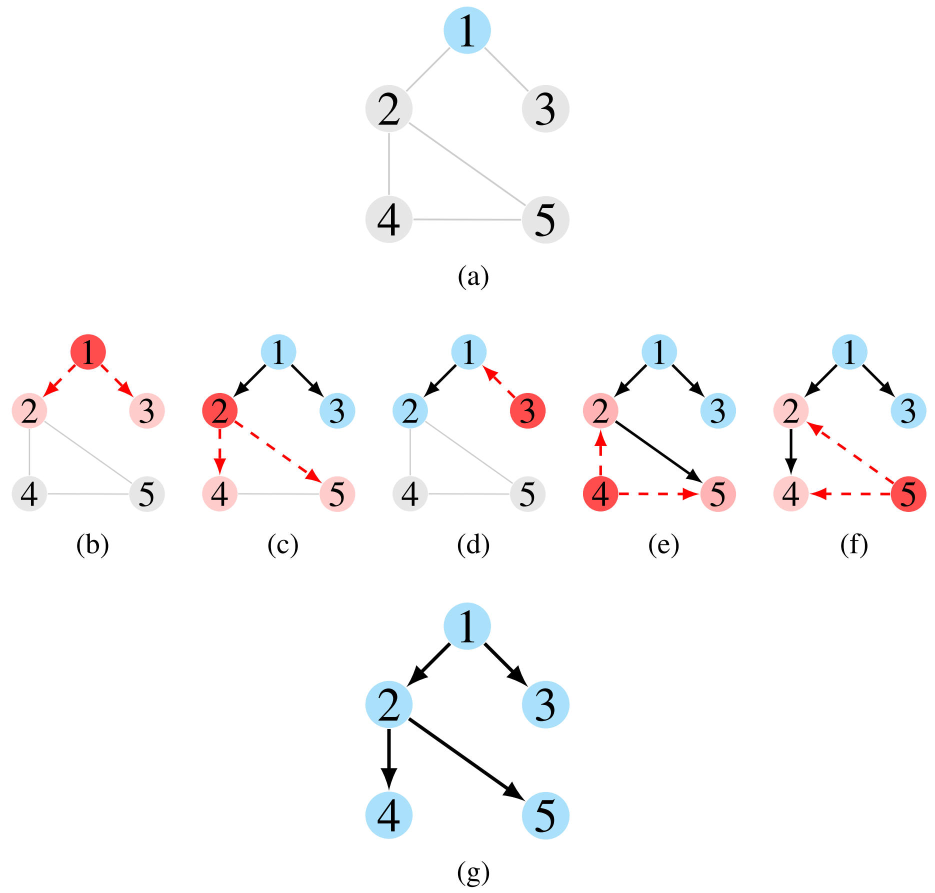


Figure 2.3: (a) Example physical topology, (b)-(f) steps for constructing RPL DAG and (g) the final valid DAG.

- Initially, the root multicasts a DIO to advertise the DAG to any nodes in the range, as shown in Figure 2.3b.
- A node that has not already joined the DAG (nodes 2 and 3 in Figure 2.3b), upon receiving the DIO, can join the DAG by selecting the sender as its *preferred parent*. It may also ignore the received DIO if it does not satisfy some criteria, *e.g.*, link quality.
- A node, which received the DIO, unicasts a DAO message to its (preferred) parent. This DAO is further propagated until it reaches the root (in case the root is not the preferred parent), the root replies with a DAO-ACK.
- After receiving the DAO-ACK, the node joins the DAG, *e.g.*, nodes 2 and 3 in Figure 2.3b.

- The newly joined node begins multicasting DIOs to allow more nodes to join the DAG through the same process (as shown in Figures 2.3c and 2.3d).
- If the node is already part of the DAG (*e.g.*, node 2 in Figure 2.3e and Figure 2.3f), it calculates the new potential rank. If the new received DIO improves its current rank, the node selects the sender as its new preferred parent and updates its rank or, otherwise, it ignores the newly received DIO, which is the case for node 2.
- The process repeats until all the reachable nodes eventually become part of the DAG and the initial route construction is complete. Figure 2.3g shows a valid DAG for the given topology in Figure 2.3a. The direction of the arrows indicates that the route traverses from a parent to a child.
- A node that is not part of RPL DAG may periodically send DIS message to solicit DAG information from its neighbour nodes. Nodes that are already part of the DAG reply with DIOs, where the same steps above begin.

A loop might be formed during the DAG constructed process, *e.g.*, a node leaves the DAG and then rejoins in its previous position. To avoid and detect loops in a routing path, RPL employs various strategies. For example, it assigns each node with a *rank* that represents the distance from the root. The rank of a child must be greater than the rank of its parent. A node selects the preferred parent according to its rank so that the best parent is the one that gives it the shortest path to the root.

2.3.2 RPL Specification

RPL maintains efficient and stable routes among different network topologies, including flat and hierarchical topologies, and supports quality of service (QoS). RPL is also flexible and can adapt while the network topology changes. It is a self-organised and self-healing protocol that utilises low-energy mechanisms when dealing with frequent node failures [35]. RPL is a proactive routing protocol that continuously updates and maintains routing information prior to the actual need for communication between nodes.

RPL Modes

RPL handles downwards routing through two different types of mode:

1. **Non-storing mode.** Here, only the root stores routing tables and includes the routing information in downwards packets. Nodes upon receiving a unicast DAO, pass it up to the root without storing the received DAO messages. Namely, each intermediate node inserts its address in the header and forwards the received DAO to its parent until reaching the root. As a result, the nodes operate with low memory usage.

2. **Storing mode.** In this mode, in contrast, the nodes require more memory usage since each node maintains a routing table. Namely, the nodes store the downwards routes of all the other nodes below them in the network. Unlike the non-storing mode, the storing mode is usually used in networks in which nodes have sufficient resources, *e.g.*, memory.

Formally verifying RPL in the storing mode increases the state space (Section 3.3) since each routing table adds additional variables, which may lead to an exponential growth of the state space for large-scale networks. Additionally, a formal verification approach explores different possible configurations, which negatively contribute to the state space explosion. Unlike the storing mode, modelling RPL in the non-storing mode does not require verifying the consistency of distributed routing tables.

RPL Objective Functions

The calculation of a node rank depends on the *objective function* employed by the RPL instance. The most widely used primary objective functions in RPL are the OF0 objective function [142] and the minimum rank with a hysteresis objective function (MRHOF) [48]. The former assigns uniform weights to links and only considers the number of hops (rank) to the root, ignoring any other performance metrics, *e.g.*, the link quality, while the latter uses various metrics, such as the minimum expected transmission count metric (ETX), to determine the best path to the root. The objective function that is used depends on the network's priorities; for example, MRHOF is preferable for lossy networks where link quality and energy consumption are critical.

RPL Trickle Timer

RPL uses an algorithm called Trickle Timer [84]. The Trickle algorithm enables nodes within unreliable networks to share information in an energy-efficient way. RPL uses this mechanism to control the traffic and, hence, the network resources. The root, or newly joined nodes, multicast DIO following a Trickle Timer. The frequency of the multicasting DIO is influenced by two values: the minimum and maximum Trickle intervals. When there is inconsistency in the network, *e.g.*, a loop is detected, the nodes reset the Trickle Timers to the minimum to increase the number of DIOs being sent. However, when the network is stable, the nodes follow the maximum interval so that the number of sending DIOs is decreased. This reduces unnecessary transmissions, thereby lowering energy consumption in the constrained devices.

RPL also uses a timer for DIS transmission. The nodes that are willing to join a DAG periodically send DIS to all the nodes within its range to reset their Trickle Timers, which increase their chance of hearing a DIO. Additionally, if changes in the network are detected, such as link failures or rank changes, the Trickle Timer resets to rapidly trigger updates. To conclude, RPL leverages the Trickle Timer to maintain efficient energy use by minimising the message transmissions when there is stability in the network, making it appropriate for networks that require low power and are prone to data loss.

RPL Common Features

RPL shares multiple features with other WSN routing protocols. For example, the construction of multiple routes between the nodes and the gateway is similar to the data fusion multipath routing protocol, which combines multiple paths between destination nodes and the source [76]. It also dynamically adjusts the path when needed, similar to the repair mechanisms of the RPL. A low energy adaptive clustering hierarchy (LEACH) [52], a self-organising routing protocol for WSNs, consists of startup and setup phases that are similar to the DIO multicasting and DIS messages in RPL. Furthermore, the parent selection in the ad hoc on-demand distance vector protocol [112], which is originally designed for mobile ad hoc networks (MANETs), is based on the hop count, which is similar to the preferred parent selection in the RPL. Moreover, RPL and routing information protocol (RIP) [121] both use a periodic update approach to share routing information with adjacent nodes to ensure an accurate network topology. These similarities, in addition to its mentioned features and popularity, make RPL a good candidate for this thesis to show how routing protocols can be formally modelled and verified, as shown later in Chapter 5.

2.4 WSNs Analysis Approaches

Since the IoT depends on networking, it is essential that the networking protocols that underpin these applications are reliable. Traditionally, testing and simulation are used to ensure system correctness and security. However, they may not be able to guarantee the reliability and security of the systems in all scenarios. Formal methods provide an accurate mathematical notation that assists designers in specifying systems' behaviour and properties that must be formally checked. The following sections cover the most used analysis techniques in IoT systems and applications.

2.4.1 Simulation and Testbeds

Since wireless sensor networks are often deployed in harsh and unattended areas, it is highly risky and infeasible in terms of money and time to design a real model for the testing of a new protocol or algorithm. Hence, other alternative mechanisms should be adopted, including adopting a mathematical analysis, using a testbed or modelling through simulation [133]. Testbeds and simulations are effective tools in the analysis and evaluation of new systems, protocols and algorithms in different design phases; namely, design, development and implementation [40].

A testbed is an infrastructure that supports the creation, modification and observation of the under-test WSN nodes, protocols and applications. Although testbeds allow for a detailed plan for real deployment, they often have a small scale and provide a single fixed topology and limited flexibility. Testbeds are also a costly option for both effort and money. Therefore, simulation is relatively cost-effective when budget and time are limited. However, combining a simulator and a testbed when testing a new system or an algorithm is the best choice in terms of acquiring the benefits from both [40].

Simulations are one of the most popular techniques, which is used to evaluate and validate WSN-related development issues and solutions. Simulations support different topologies on a large scale and allow for a rapid and repeated experiment that is used to try new ideas [40]. They do not require real-world deployments when studying new approaches and techniques and also when evaluating networks that contain hundreds or thousands of nodes. Furthermore, simulations assist in analysing network performance metrics such as delay and lifetime. There exist two main types of tools for testing WSNs: simulators and emulators. The simulators are concerned with a high-level analysis, *e.g.*, data aggregation and network topology, since they simulate the behaviour of a system. Emulators, on the other hand, are used when the focus is on low-level details, such as sensor algorithms, because they replicate the functionality of applications to mimic real-world systems [40].

Although the simulation does not reflect the actual execution, it creates an abstract model to analyse the behaviour of the application under specific conditions. Different types of simulation exist, such as continuous simulation, discrete event simulation (DES) and stochastic simulation. For wireless sensor networks, a DES is preferred because it facilitates simulating many jobs or activities that occur at various nodes at a discrete interval time. Simulation assists a developer in understanding and analysing the proposed system in various scenarios. It also enables a visibility of the actual behaviour [133].

Simulation provides a clear understanding and analysis of the system by conducting a large number of tests under different scenarios in the same environment with some changes for the simulation parameters when needed. That is, multiple what-if scenarios can be tested, hence, enabling the decision-makers to understand different alternatives. The simulation does not require a proper deployment of the hardware infrastructure, so it can be seen as a time-efficient approach since it does not take as long as a real implementation. Different areas utilise different simulations for the explored applications. For example, stochastic simulation is used in applications where there are random changes in variables with probabilities changed, while a deterministic simulation is performed for the systems that have deterministic behaviour, *i.e.*, no degree of randomness occurs [134].

To design a simulation environment, a number of steps are conducted. First, determine the purpose of the simulation and the network topology of the abstracted model of the system. Then, set the values of the simulation parameters and decide on the simulation assumptions. The simulation is now ready to start, depending on the performance metrics. The graphical presentation showing the simulation outcomes is presented. Finally, the simulation results are analysed and the best alternatives are found [134].

Several simulations are specially designed for WSNs. The following are some examples:

- **Cooja [27]**

Cooja is a popular emulator in the field of WSNs. It has been used in numerous research articles, *e.g.* [144, 64, 3, 132]. It is part of the Contiki operating system [106] and is used to simulate the radio environment. Contiki supports a number of standards and protocols, including RPL; hence, simulated WSN motes in Cooja have access to most of them. Cooja emulates node firmware at the instruction level that is executed in a simulated wireless communication environment. Cooja offers simultaneous simulation at different levels; namely, network level, operating system level and machine code instruction set level. Cooja has been implemented in Java, which facilitates its extensibility for users. Additionally, Cooja features a graphical user interface that supports its usability and, hence, facilitates its use in several protocol-related studies.

- **J-Sim (Java-simulator) [136, 103]**

J-Sim is a general purpose open source component-based development in Java for WSNs. Since it is implemented in Java, along with its autonomous component architecture, J-Sim is a platform-independent, extensible and reusable environment. That is, a component can be designed, implemented and simulated independently in J-Sim. It offers a framework for WSN simulation where three WSN protocols are implemented; namely, geographic routing, directed diffusion and localization.

- **Ns-3 [103, 40]**

Ns-3 is a discrete-event simulator for research and educational use. It was developed to enhance communication network research. C++ is employed to develop NS-3 programs, with an option for Python bindings. Multiple WSN models are readily available in NS-3, such as 802.15.4 and RPL.

- **TinyOS simulator (TOSSIM) [85, 35]**

TOSSIM is a discrete-event simulator for TinyOS wireless sensor networks. TinyOS is an operating system specifically designed for sensor networks that has a component-based programming model. TOSSIM allows users to debug, test and analyse algorithms on their PCs because they are able to compile them into the TOSSIM framework. It does not offer a graphical user interface (GUI) and it partially supports RPL.

- **WSim/WSNet [103, 35]**

WSim/WSNet is an event-driven WSN simulator that is developed to debug and evaluate the performance of embedded WSN applications. It is compatible with different embedded operating systems, such as TinyOS and ContikiOS. It is a command-line-based simulator that does not support a GUI. A RPL module was implemented in the WSNet simulator. It consists of the route construction, the distance to the root computation and the packet transmission. However, it only considers the upwards routes.

In this thesis, we opt to use Cooja to conduct our comparison for analysing RPL protocol using simulation and formal methods (Chapter 6) because of its native support for RPL, which makes it the most widely used simulator for RPL routing protocol related research. Using Google Scholar, the term ("COOJA" AND "RPL") returns more than 2000 works in the last five years. However, using the same approach for the other highlighted simulators returns a maximum of 503 for the Ns-3 simulator. A recent review study [4] found that Cooja has been used by 71% of the 127 reviewed studies. Additionally, Cooja offers a user-friendly interface with rich visualisation capabilities. Due to its popularity, Cooja enjoys large community support with its extensive documentation and tutorials.

2.4.2 Formal Verification

Despite the advantages highlighted for using simulations to verify systems and protocols using simulations, simulations do not provide full coverage of all potential outcomes [46, 47]. Simulations execute several replications and carry out statistical analysis to assess system behaviour across various runs for a given scenario. In critical systems, where every possible outcome must be validated, another verification approach is essential.

Formal verification approaches, in contrast, perform an exhaustive check for all the possible combinations, providing full coverage of the state space, which leads to the discovery of hidden bugs and errors. Unlike simulations, which validate implementations, formal verification proves that a given system satisfies its specification. The next chapter provides more details on formal verification approaches, including a comparison between simulation and formal verification approaches.

2.5 Summary

This chapter has emphasised the significance of WSNs in supporting IoT. Hence, it is important to verify them. The chapter begins with a general background on WSNs along with one of their key aspects, *i.e.*, routing is covered in Sections 2.1 and 2.2. A description of RPL, a widely used routing protocol, is presented in Section 2.3. Due to the popularity and features of RPL, this thesis uses it to illustrate how routing protocols are formally modelled and verified, as we will see later in Chapter 5. Section 2.4 has illustrated the main analysis approaches for WSNs: simulation and formal verification.

Although simulation is widely used as a reliable technique to validate protocols, analysing WSN routing protocols requires a more comprehensive approach due to its importance in WSNs. The next chapter covers a formal verification approach as another analysis method for WSNs and compares its features with simulation.

Chapter 3

Formal Methods

This chapter begins by providing a general definition for formal methods (Section 3.1). Then, Section 3.2 elaborates on the different approaches for formal verification, while Section 3.3 discusses state-space explosion, a well-known issue in formal verification. A comparison between the application of formal verification and simulation for the analysis of protocols is given in Section 3.4. The chapter presents a review of existing related works that apply formal verification to WSNs (Section 3.5), and Section 3.6 discusses the necessary features of the formal technique used to formally verify routing protocols while Section 3.7 summarises the chapter.

3.1 Formal Methods

Formal methods are mathematical techniques that are used to model and reason about systems. They are employed to check completeness, correctness and inconsistency. By formally analysing systems under different scenarios, one can detect vulnerabilities, attacks and errors in the early stages of the design. Formal methods provide abstraction for a system, *i.e.*, allows a high-level representation, which assists in a rigorous analysis. They help in understanding, analysing and predicting the behaviour of the system. Formal methods allow for overall quality improvement for a system by providing efficient test scenarios, which also guarantee that a system satisfies its requirements. They facilitate the maintainability and reusability of systems by providing a clear and constant description of the system's behaviour.

Formal modelling relates to a family of approaches, strategies and tools that use mathematical or logical methods to create abstract representations (models) of intended systems. The literature is rich in research articles that present different types of formal modelling language, techniques and tools. As given in the following, the latter are divided into several categories. Formal specifications and languages, such as TLA+ [80], Event-B [1] and Alloy [63], use *mathematical concepts* to unambiguously define the behaviour and structure of systems. The other category provides formalisms that are based on *process algebraic*, which assists in reasoning about concurrent communicating systems, such as π calculus [98]) and calculus of communicating systems (CCSs) [96]. Other models that use *rewriting logic*, such as Maude [32], to model and analyse system behaviour via rule-based transformation provide support for modelling hierarchical structures and state transitions. There also exists *graphical and mathematical methods* that are based on graph theory to provide a graphical description of states and transitions, which model workflows and processes in concurrent systems, *e.g.*, Petri nets [118] and graph transformation systems (GTSs) [39]. Once a model for a considered system, protocol or algorithm is constructed using an appropriate modelling language or tool, it is then formally verified using an appropriate technique.

In order to check different parts of the system, such as the functional correctness of implementations, the formal verification procedure uses a diverse set of mathematical and logical methods to ensure the correctness of the designs and to provide quantitative statements about the safety and security of a given system [55]. There are four main areas in which formal verification methods could be applied to protocols that are used in the IoT environment [55]. These areas are given in the following. First, functional tests that are performed to check, *e.g.*, the protocol stack and the time needed for encryption key updates. Second, proposed schemes or extended protocols that are checked by using formal methods. Third, the application of formal verification methods in order to verify security properties. The last one focuses on checking the implementations of protocols due to the possibility of misunderstandings during the translation of the protocol into code.

3.2 Formal Verification Approaches

Formal verification is an established technique that finds potential vulnerabilities and weaknesses. Although they are usually used in the design phase to provide security guarantees, unlike simulation, formal approaches can be used throughout the protocol development lifecycle. Formal verification and validation approaches are used to ensure the reliability and security and safety of IoT applications. The process of formally verifying a system is generally described as follows: first, it studies the specification, *i.e.*, a formal description of a system, in detail. Then, it defines a model based on the given specification using a modelling language or tool. The next step is to translate the defined model into the input language of the model checking tool. Finally, it checks the results of the formal verification and, based on these results, recommendations are probably defined for a set of standard changes.

To demonstrate the formal verification concepts and terms, a traffic light system is now used as an example. The traffic light system consists of three main *states*: green, yellow and red. Here, “Green”, “Yellow” and “Red” denote when vehicles can go, prepare to stop and must stop, respectively. *Transitions* indicate how the system moves from one state to another. The system can be extended to include, *e.g.*, the pedestrian “Wait” state where pedestrian signals manage the crossings; it can also include all directions, *i.e.*, north-south, east-west, etc., to fully

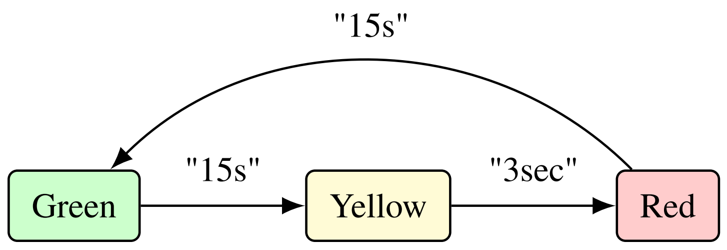


Figure 3.1: Example of a traffic light transition system.

specify the system. The transitions can occur after a set time, *e.g.*, the system moves from "Green" to "Yellow" after 15 s or upon the accruing of an event, *e.g.*, after a pedestrian presses the pedestrian signal. The *state space* encompasses every potential state and transition within the system. Figure 3.1 shows a simple state transition system for the traffic lights. A formal method tool is then utilised to automatically verify the model of a system in its model state space according to a given specification against some properties.

There exist two categories of these automated proofs: model checkers and theorem provers [19]. *Model checkers* essentially create comprehensive tests that include all possible scenarios. Therefore, a model-checking approach provides a proof of correctness for every single sequence of events that might occur. It usually does not require an expert because it is fully automated. In contrast, *theorem provers* usually require human expertise, *i.e.*, someone who provides the characteristics of the design and specification, such as algebraic constraints or theorems, in order to guide a proof of correctness [55]. However, there is a third approach that uses pen and paper proofs to manually perform mathematical reasoning about an abstracted system.

3.2.1 Model Checking

Model checking is a model-based formal method techniques, in which each state of a model can be comprehensively checked to prove that a system satisfies a set of properties [151]. Various model checkers have been developed to formally analyse systems and applications, *e.g.* [92, 125, 143]. Model checking is based on temporal logic, in which a model is a transition system and a property is a formula that is specified in temporal logic. That is, model checking verifies systems and applications in terms of a property, *i.e.*, a formula of temporal logic, that satisfies the considered system requirements. To formally check the model, a model checker is run using the model and the property as the input. The result is "True" when the model satisfies the property and "False" if it does not. Many model checkers generate a trace that causes undesired results. The model can be abstracted so that many features are ignored, but none of them should relate to the checked property. For example, in the traffic light system, the yellow state and exact timing details can be abstracted away for all the traffic directions when we need to verify that a direction eventually has a green light. That is in the abstracted model, the states are only "North-South Green" and "North-South Red".

Temporal Logics

When the temporal ordering of events or states in a system over time is critical, a number of families of formal logics called temporal logics can be used to reason about multiple possible sequences of both future and past events. They are used to express interesting properties, as logical expressions that are readable for model checkers.

They can be classified according to the temporal logic view of the time, such as linear time logic (LTL) and branching time logic called computation tree logic (CTL). The former is a set of paths, each path is an instance of a sequence of time, whereas the latter is a logic tree with a root that represents the current time and branches that are the future states. The branching logic represents non-deterministic behaviour more explicit than the linear one [60]. Unlike LTL properties, which check whether a property holds for all the paths, CTL checks the existence of a path that satisfies the property. LTL uses the path formulae, where **G** (globally - all future states), **F** (eventually - some future states), **X** (next) and **U** (until) are used. CTL additionally uses quantifiers for the paths, *i.e.*, **A** (all paths) and **E** (there exists a path), to explicitly quantify the paths. For example, $\mathbf{E}\,[\,\mathbf{F}\,\phi\,]$ means there exists a path that eventually reaches a state satisfying ϕ. If the checked property is essentially holding for all the paths, we specify it as $\mathbf{A}\,[\,\mathbf{F}\,\phi\,]$.

To specify and verify the properties of probabilistic systems, CTL is extended into probabilistic computation tree logic (PCTL). The probability quantifier **P** is used with path quantifiers of the CTL to check the probability of reaching a specific state. For example, $\mathbf{P} \geq 0.9\,[\,\mathbf{F}\,\phi\,]$ means, with a probability of more than 90%, the system eventually reaches a state satisfying ϕ. We will show more examples of temporal logics which are used later in this thesis Chapters 6 and 7.

Model Checkers

Different model checkers serve different aspects. The following describes some of the model checkers.

- **Explicit-State Model Checkers**

In this variant of model checkers, the system's state space is explicitly explored to ensure that all behaviours satisfy a given specification. A counterexample for the violation properties is generated, which can be further investigated. SPIN [57], for example, is a popular explicit state model checker that is used to verify a wide range of applications, *e.g.*, distributed systems and communication protocols. To formally check a system in SPIN, the system behaviour is specified using a specification language (Promela), whose syntax and semantics are similar to C\C++. The interesting properties are written in linear temporal logic (LTL) and a model checker performs a comprehensive state exploration. The violation properties can be investigated using the SPIN interactive simulator. SPIN features a graphical user interface that facilitates running simulations. SPIN has been used in different network applications [125, 115]. Unlike explicit state model checkers, symbolic model checking [23] represents a collection of states that compactly (instead of systematically) check each one. Therefore, it can more efficiently handle large state space such as STORM [36], which supports both approaches. However, parameterised model checkers such as Cubicle [33] abstract the behaviour of individual component to verify protocols or systems with an arbitrary number of components by verifying properties across varying system sizes, allowing for scalable and flexible analysis. When dealing with very large or highly complex systems, this approach can be computationally expensive and may suffer from state space explosion. Another challenge is that not all protocols or systems can be easily abstracted into parameterised models.

- **Timed Model Checkers**

Another type of model checker focuses on handling systems with time constraints, such as real-time systems. That is, a model checker that verifies that a system will meet timing deadlines and critical events will occur within strict timing constraints. UPPAAL [81] is an example of such a model checker. It is based on the theory of timed automata and utilises CTL to specify properties. Due to its intuitive graphical interface and since its focus is on speed and usability, UPPAAL has gained wide adoption in both academia, as a teaching tool for model checking, and for verifying applications *e.g.* [68, 117]. Generally, timed model checkers inherently lack support for probabilistic behaviour, unless enhanced with supplementary frameworks.

- **Probabilistic Model Checkers**

This type of model checker is used to check models with randomness, uncertainties or probabilities in their transitions or states. They verify properties related to the likelihood of certain system states. PRISM [77] is one of the most widely used model checker. It consists of two parts: a modelling language and a model checker. PRISM is a free and open-source probabilistic model checker developed at Birmingham and Oxford Universities since 1999. PRISM has been used in many application domains, such as communication and multimedia protocols, security protocols and biological systems. The analysis process is fully automated and provides visualisation of quantitative results [77]. It uses a modelling language in order to describe probabilistic state transition systems [41]. Then, PRISM uses a model checker to formally verify the formal model.

PRISM supports a number of Markov chain variants including discrete-time Markov chain (DTMC), continuous-time Markov chain (CTMC), Markov decision process (MDP) and probabilistic timed automaton (PTAs)(Section 4.1). It also provides verification abilities for properties that are expressed in different temporal logics, including PCTL, LTL and CSL. In cases where the model is non-probabilistic, it is still supported by PRISM by treating, *e.g.*, a transition system 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) [30]. PRISM lacks in abstracting, which results in the model becoming more complicated as the system size increases. Additionally, modelling dynamic behaviour is a challenging task in PRISM. Using PRISM, we can make use of a model checking approach to formally verify the bigraphical models presented in this thesis because of its compatibility with BigraphER, *i.e.*, a rewriting engine tool we also use to implement our models. PRISM supports different variants of bigraphs models, which include non-probabilistic and probabilistic models, as well as the underlying MDP and PTA models (Chapters 6 and 7).

Despite the fact that the model checking approach is useful since it can automatically find property violations, it may not be able to deal with a state space explosion (Section 3.3). This is because it exhaustively verifies the transition system. Additionally, some model checkers lack the capability of handling infinite systems.

3.2.2 Theorem Proving

In contrast to model checkers, the theorem proving approach [102] is another formal verification method that can be used to verify infinite systems by verifying critical properties of a system specified by using mathematical logic. The theorem proving approach was essentially developed to prove mathematics theorems by computer programs. Over time, it has evolved and has been used in modelling and reasoning about complex systems. Similar to model checking, theorem proving verifies a property under concern and provides a definitive result. Theorem provers, which are known as mechanical reasoning systems, can be classified into automated theorem provers (ATPs) and interactive theorem provers (ITPs). The former proves the goal via dealing with the development of automated computer programs. The latter needs human interaction with the computer to prove the goal. Theorem provers based on logic are categorised as propositional logic theorem provers, first-order logic (FOL) theorem provers and higher-order logic (HOL) theorem provers. Some verification tools serve general purposes, *e.g.*, the HOL theorem prover based on higher order logic [61], while other theorem provers are specifically designed to support protocol design and analysis, such as ProVerif [21] and Tamarin [95].

These verification tools are popular in the field of network protocol verification. For example, ProVerif is also used to verify proposed enhancements to an existing WSN authentication protocol that suffers from some attacks and violates secrecy and anonymity [153]. ProVerif is used to simulate and verify an improved symmetric key-based authentication protocol for IoT-based WSNs [45]. Tamarin is another widely used tool for protocol verification, especially for security properties. It is employed to verify a proposal to enhance the reactive-greedy-reactive (RGR) routing protocol for unmanned aeronautical ad hoc networks (UAANETs) [100]. Tamarin is also used to verify a proposed formal definition of flow integrity by analysing secure modes of industrial protocols [38]. The work in [19] shows that it is feasible to use automatic verification tools to verify the core properties of the Internet Engineering Task Force (IETF) standard or draft specifications for routing protocols. For a predefined network, the SPIN model checker is employed to simulate and verify the protocol. However, for an arbitrary network, HOL (the interactive theorem prover) is utilised to verify the general mathematical properties of the protocol with more human intervention. Their results reveal that using formal analysis to verify routing protocols is possible with acceptable effort and speed. Therefore, their approach is claimed to be an effective complement to other techniques, *e.g.*, manual proof and testing, since it can identify a wide range of cases and automatically check them.

Unlike model checkers, theorem provers do not suffer from state space explosion because they use an abstraction on protocols with unbounded sessions. However, this abstraction may lead to reports of false attacks when analysing protocols with global states. In addition, due to the fact that some theorems are undecidable, Tamarin and ProVerif, for example, may not terminate at all. Furthermore, theorem provers are not efficiently sufficient in formalisation for the majority of the mathematical community, as they still suffer from a shortage of libraries that support proofs automation and also the interfaces need improvement [102].

Another limitation is that they lose automation since supporting protocols with no limitations of states require human intervention by providing auxiliary lemmata. This can be challenging even for experts [156, 26]. According to [86], users can apply theorem proving on all valid network topologies, but dealing with complex networks is a challenging task since it relies significantly on the users' familiarity with the system. Using the theorem proving approach presents difficulties due to the complexity of the underlying logic. Moreover, theorem proving does not provide a counter-example when verifying a property. In addition, theorem proving traditionally requires human involvement and, hence, it is slow and prone to errors, despite the best efforts of automating (*e.g.* [157]).

To this end, model checkers are to be applied when verifying protocols on small networks. This is due to the state-space explosion. In addition, as it has fully automated support, it attracts non-experts to use them. However, on the other hand, theorem provers can be utilised on large networks since they have the ability to verify unbounded networks [26].

3.2.3 Theoretical Analysis

Theoretical analysis is another formal verification technique, in which a mathematical examination is carried out on a system to verify that it satisfies predefined properties. To achieve this task, an abstract model that represents key behaviour and possible states of the system should be created first and, then, the essential properties that the system must meet are defined. Compared with model checking and theorem proving, theoretical analysis manually proves a system that is represented by mathematical equations using logic and algebra. Furthermore, while model checking exhaustively verifies the state space to provide a definite answer, theoretical analysis provides general insights into the properties of interest.

Theoretical analysis is utilised by various works that are performed to verify various aspects of WSNs. For example, [88] lists some benefits that are obtained by theoretical analysis of the lifetime and energy hole in cluster-based WSNs. They state that theoretical analysis allows them to achieve the expected lifetime before deploying the network, know the principal components that affect the network lifetime and provide them with awareness of the weaknesses of the network before deployment. Other work adopts integral geometric theory to theoretically analyse the coverage issue in a UWSN, an unmanned aerial vehicle-based WSN. They then verified the accuracy of their proposed theoretical analyses via MATLAB simulations [137].

Radio frequency identification (RFID), which is a technology that uses radio waves to wirelessly transmit the identity of an entity as a unique serial number, is theoretically analysed to improve security and privacy issues [163]. The latter work uses three security protocols to study and compare several RFID criteria. The results reveal that theoretically analysing the protocol assists in finding the level of security and privacy. Theoretical analysis can also be performed with the help of proof assistant tools, such as Coq [59] and Isabelle/HOL [104], by formalising complicated structures and automating reasoning.

Similarly, a static analysis approach can be utilised to verify the model without implementing it, which is similar to examining the code of the software program without actually executing it. Static analysis is a formal approach that can be employed to analyse a system early in the development cycle, which leads to improved system reliability, security and maintainability. It can also be leveraged to ensure the correctness of the constructed model regardless of its execution, as we will see in Chapter 6 where we validate the models regardless of the network topology.

3.3 State Space Explosion

Despite the benefits of verifying systems and applications by utilising the model checking approach, model checking faces the issue of explosive growth of the state space. State-space explosion is a well-known issue in the field of formal verification. The size of a system and the space state explosion are directly proportional. For instance, as we add more directions to the traffic light system example, more states will be generated, leading to a larger state space. This is due to the fact that the model checking approach explores all possible states and all possible transitions between these states in order to verify that a specified property holds. That is, for a system consisting of n intersections, each having m states, the state space would be m^n states, which all need to be explored.

Due to the fact that the state space issue cannot be avoided in some cases [31], several techniques have been developed to alleviate the issue. Abstraction techniques, for example, rely on simplifying the model to reduce the complexity of the system while preserving the essential properties. Another technique, known as symbolic techniques, compacts a set of states and transitions instead of representing them individually, resulting in a smaller state space. However, reduction techniques focus on eliminating unnecessary states and paths. Symmetry reduction identifies and collapses equivalent states, while partial-order reduction avoids exploring the interleaving of an independent transition that is seen as unnecessary [31, 155]. Additionally, when verifying a system that has uncertainty or large state spaces, the results can be approximated with probabilistic guarantees using statistical model checking [83].

3.3.1 State Space Explosion Mitigation Applications

Generally, model checkers utilise various mitigation techniques to control the state space problem, For example, UPPAAL employs different techniques, such as partial-order reduction [17] and symmetry reduction [53], to handle state space explosion. However, modellers usually apply different strategies and techniques to mitigate the resulting transition systems.

Abstracting systems and protocol specifications to only model the core functionalities is the technique typically used. For example, using Maude [32], *i.e.*, a rewriting engine that provides a variety of analysis techniques such as LTL model checking, message collision has not been considered when modelling the ad-hoc on-demand distance vector (AODV) routing protocol, of which RPL is an instance [89]. The authors conclude by suggesting that statistical model checking techniques and/or abstraction techniques for mobile ad hoc networks (MANETs) should be developed. Another work uses algebra of wireless networks (AWNs) to provide a formal specification of the main functionality of the AODV routing protocol [47]. The AODV model covers the main components of the protocol while abstracting timing aspects and optional features. The work reasons about key features of the AODV, *i.e.*, loop freedom and route correctness, which are used as examples.

Another work, however, considers various assumptions when implementing the correctness of distributed systems and the security properties of complex systems using the temporal logic of actions (TLA+) that is verified through TLC, as is found in [37]. It assumes that the authorised user is allowed to have limited access to the intended services. The attackers are also restricted to performing limited attacks for each detected intelligence. The message queuing telemetry transport (MQTT) protocol is modelled while imposing various assumptions to restrict the state space, *i.e.*, by assuming a limited number of clients and considering a single topic, alongside bounded packet identifiers [120]. It also employs an incremental model checking approach based on the different stages of the protocol.

Although the nature of some of the systems makes the state-space explosion unavoidable, such as routing protocols where, *e.g.*, nodes are continually joining and leaving and there are a number of possible routes, different strategies can be considered. One work tackles this by applying a symmetry reduction technique. It focuses on the shape of the topology. That is, if the shape of the topology of the current state seems to be the same as the one represented by the searched state, then both states are considered equivalent, thus nodes are replaced [72]. Other systems that suffer from state space explosion are real-time systems that often deal with implicit clock synchronisation. As an example, the work of [17] attempts to reduce the generated state space by employing a partial-order reduction technique. The proposed work permits clocks from different processes to advance independently; then, these clocks can be resyncronised again if their processes communicate with each other.

Given that each technique has a different approach, the selection of the appropriate approach is influenced by many factors. For example, if a system under consideration can be simplified while maintaining its core functionality, abstraction techniques can be applied here. However, when necessary, multiple approaches can be combined. For example, if a system has complex parallel processes, abstraction techniques can be applied: first to reduce the state space and then to use reduction techniques to minimise the state space further. In this thesis, model abstracting is applied as a first step with various assumptions used. Increment modelling, which is supported by bigraphs, is also considered, as we will see later in Chapter 5.

3.4 Formal Verification vs. Simulation

Formal verification approaches have been widely used for rigorous system analysis and also for the design and analysis of IoT protocols. Several works emphasise the benefits of using formal methods to analyse different applications, protocols and systems over simulations. The following sections provide an overview of these findings.

3.4.1 The Full Coverage

In contrast to the simulation approach, the formal verification approach provides more coverage. The formal verification approach has been shown to have 100% coverage, whereas simulation provides 94% coverage [46]. The latter study sets different parameters for their comparison using the Manchester encoding for the MIL 1553-b protocol as a case study. This leads to more reliable results since formal verification guarantees the absence of bugs that cannot be guaranteed with simulation for small designs where the state space is finite. Another study [47] agrees with these findings using algebra for wireless networks, which is a process of algebra specifically designed for the modelling of mobile ad hoc networks and wireless mesh network protocols. It provides a complete and accurate model of the core functionality of the ad hoc on-demand distance vector routing protocol. The study claims that other protocol evaluation approaches, such as simulation and testbed experiments, do not provide robust answers regarding protocol behaviour due to their limited coverage of network scenarios. They suggested that using model checkers, such as UPPAAL, ensures the discovery of potential errors in an early protocol development stage.

3.4.2 Earlier Application

Verification tools can help validate protocol specifications, as concluded by [19]. The latter work shows the value of the existence of protocol specifications since they demonstrate that analysing protocols through their standards can complement other analysis approaches, such as simulation. Additionally, due to the use of English prose in specifying the AODV specification, there is an ambiguity that may affect implementations, which might be the case for other IETF specifications [47]. Recently, formal verification was involved in the early stage of protocol design [73].

Quint[1], a newly developed specification language, is used to write a formal specification for the ChonkyBFT algorithm, which is a new Byzantine fault-tolerant protocol for distributed systems [164]. The model checking results show that there are some small inconsistencies in the informal specification and also a few missing message validation tests. Their approach also results in the identification of a few problematic data structures. Despite the slowdown of the model checker, the work finds that formal verification has more coverage than simulation [73].

3.4.3 Finding Bugs

While some works verify that a protocol works as described in its specification, *e.g.*, [141], some others result in the discovery of errors and bugs when applying formal verification. Detecting and resolving bugs before deploying the protocol leads to adding more guarantees to the validity of systems [101]. The latter work develops a mathematical model of the adaptive data rate (ADR) protocol for LoRaWAN networks to develop multiple specifications in the natural language of the protocol. The last specification is the one that is converted into an algebraic notation for formal checking. They use Event-B to formally model the protocol server side and verify its properties with Event-B invariants. Alloy [63], a modelling and analysis language, is used to model the Android permission protocol that provides a fully automatic analysis [13]. This leads to the identification of three types of vulnerability that can allow unauthorised access. Additionally, many secured routing protocols for ad-hoc networks contain design flaws and are, therefore, vulnerable to attacks [140]. The latter work finds that, by initially assuming an arbitrary topology with a strong attacker model, using a backward searching approach to verify an invalid route is supposed to be accepted at the end of the route discovery. That is, in contrast to ProVerif, which requires a predefined topology and protocol specification and is translated into logical rules for automatic verification using a forward search method. The work also developed a software tool based on the proposed approach that successfully finds attack scenarios in well-known routing protocols.

3.4.4 Other Benefits

Unlike simulation that simulates dynamic environmental changes, the use of probabilistic models in protocol analysis permits quantitative analysis for the system with the dynamical environment effects that cause WSN failure [151]. In addition, simulation requires adjustment and redeployment when the detection results do not satisfy the requirements. Utilising Petri nets, for example, to construct a trust-based formal model (TFM), the fault detection process and faults can be described and checked without simulating and running a WSN. The proposed TFM allows for quick modification of existing designs and, therefore, a reanalysis of updated designs regardless of network size [150]. It also found that simulation results require the setup of an environment to

[1]https://github.com/informalsystems/quint

obtain the required results. However, formal verification results can be extracted from the initial phase [46]. Furthermore, there is a small gap between the accuracy of the real-field deployment and simulation [131]. The latter work performs evaluation and validation of the results that are found from a wireless sensor network deployment with those obtained from ns2, which is an open-source simulator. The study simulates and implements the agent-based S-MAC protocol on the ESB sensor node. Here, there needs to be a trade-off between time, investment and the required precision of the results.

3.4.5 Complimentary Advantages

There exist a number of studies that investigate using *multiple approaches* to verify systems for different reasons. For example, although different IoT protocols such as Zigbee, Bluetooth, 6LoWPAN and 5G are assessed against various security properties, which are availability, confidentiality, integrity, authenticity and accuracy, it has been found that it is generally not possible to verify the whole protocol [55]. Thus, critical parts for a detailed consideration of the specification or the standard should be identified to be modelled and then transferred to the input language of the selected model checker. Adjustments to the protocol model might be required due to the limitations of the selected tool, which may result in a weaker result. To address this issue, a combination of several tools should be applied to improve the overall result. Due to the different input languages of the tools, a different input model has to be created; translators or tools for assisted translation might be helpful here.

Another reason is that the hidden bugs found in simulations are successfully identified by formal verification, as shown in [139]. During the verification step, through the simulation of a sensor network communication that senses temperature, the base station does not receive the temperature data of all the nodes. Utilising the UPAAL model checker, it is found that the FIFO used in the nodes was the cause of the bug. The work formally modelled and verified the FIFO of the motes until it satisfied the specified property.

Furthermore, utilising multiple methods to verify critical IoT systems usually has complementary characteristics. For example, coloured Petri nets (CPNs) are used to model and verify MQTT protocol logic that covers all three levels of quality of service that MQTT provides for delivering messages between clients and servers: *at most once*, *at least once*, and *exactly once* [120]. They employ both simulations, to test the appropriate operation of the model, and exhaustive validation by exploring the state space. This resulted in the discovery of several issues related to the implementation of quality of service levels that may lead to interoperability problems between implementations. Similarly, the correctness of a new version of the SACK TCP protocol is proven by formally modelling it and using simulation to test it in a relatively real environment [159].

To this end, both formal verification and simulation are useful and important techniques for validating and verifying protocols. However, the nature of the protocols and the focus of the study influence the approach to use. For example, when the focus is to gain a quick insight into how a protocol behaves under various network conditions, simulation perhaps is the best choice. However, when correctness is required, formal verification guarantees the absence of bugs as it applies an exhaustive verification. However, the best practice is to use both to achieve the complementary benefits.

3.5 Formal Methods for WSNs Verification

Modelling a WSN is a challenging task, as it is modelling a distributed system and an embedded system at the same time. Additionally, WSNs imply many constraints that affect the network, such as energy and memory limitations. The work in [108] classifies the current WSN models as follows. The first group relates to the modelling of the node level, *e.g.*, energy management and coverage. The second is concerned with the network itself, *e.g.*, connectivity and routing protocols, while the third category focuses on application metrics, *e.g.*, quality of service and reliability. The last one considers IoT modelling strategies and simulation tools.

3.5.1 Properties of WSNs

Several works have modelled different aspects of the WSNs using formal specifications and formalisms. Survivability, which is the ability of a WSNs to continue to function despite the existence of failures or other challenging conditions, is formally evaluated by modelling the behaviour of the WSNs in the environment of the PRISM model checker using CTMCs [114]. The work defines four measurements, which are the frequency of failures, the data loss, the delay and the compromised data. Three failure types are considered: node, link and attack failures, communication faults and black hole attacks. For the latter, a compromised node simply drops all the packets it received to cause network disruption. This approach is flexible and can be used for different types of networks considering their own specifications.

The energy efficiency in RAEED with load balancing (RAEED-LB), a proposed WSN protocol, is formally verified to evaluate the performance of the protocol with respect to energy as a crucial factor. RAEED-LB is a new energy-efficient version of the RAEED protocol that is designed to prevent denial of service (DoS) attacks. UPPAAL is used to check the minimum lifetime of each protocol. The results show that RAEED-LB obtains a network lifetime that is 10%–30% greater than RAEED [68].

The security of WSNs is investigated by combining both formal modelling and simulation to evaluate the effects of attacks and the effectiveness of possible security mechanisms in WSNs [18]. By formally building the abstract model of the system with various attacks, and then measuring the effects of these attacks on the network parameters of interest, such as energy consumption through simulations, WSN designers can obtain the required knowledge in terms of design flaws and security-related issues, thus identifying appropriate solutions.

3.5.2 Protocols of WSNs

Different WSN protocols are modelled and analysed using formal method approaches. Three complementary protocols – namely, TinySec, LEAP and TinyPK – are formally analysed using model checking [143]. The role of these protocols, respectively, are as follows: authenticating and encrypting messages, covering the key distribution mechanism and permitting one to establish an authenticated conversation with an external third party. The protocols are modelled in HLPSL, a high-level formal language, and verified using the automated validation of internet security protocol and applications (AVISPA) model checking tool. The authenticity and confidentiality of messages are checked, which results in the detection of man-in-the-middle and type-flaw attacks. The work consequently supports the solutions.

WSNs use the carrier sense multiple access with collision avoidance (CSMA/CA) protocol to minimise collisions and ensure smooth network operation by efficiently managing access to the shared communication channel. Using hierarchical timed coloured Petri nets (HTCPNs), which have the ability to model time constraints inherent to the protocol, the study of [165] models CSMA/CA and analyses the models with properties that are not verified in most works that exist in the literature; this is achieved by the employment of the CPN-Tools. The hierarchical approach allows them to connect small models to build a large network model. In addition, the work conducted qualitative and quantitative verification for their model with a random number of arriving packets and a variable number of nodes. A collision resolution algorithm for WSNs (2CS-WSN) is formally analysed using the PRISM model checker, which is motivated by the inability of general simulators to evaluate the expected collision resolution time and properties [92]. A SPIN model checker is used to verify the correctness properties of an encryption scheme for WSNs that describes its behaviour written in Petri nets using Promela constructs [125]. The work illustrates that eventual errors can be detected at an early stage. Furthermore, a review of the generated counterexample enables the correction of the detected errors.

3.6 Discussion

The techniques applied to the analyse and verify routing protocols must consider the nature of such protocols, which are as follows. Routing protocols are capable of handling dynamic changes in the topology and node or link failures since they are usually applied to dynamic networks. Routing protocols should also handle the simultaneous execution of numerous simple processes. Furthermore, the protocols respond to the real-time actions. In addition, the study is considered accurate if it is successfully generalisable to any number of nodes [19].

To investigate the feasibility of formally modelling such a protocol, this thesis proposes bigraphical reactive systems (BRSs) [97]. It assesses BRSs based on their ability to satisfy the following characteristics:

- Its ability to relate entities both spatially and with non-local connectivity since sensor networks 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.
- Its ability to allow for complex user-defined rules. This is to enable modelling of both locality and connectivity in a single formalism, which is unlike fixed rules that are found in π-calculus for example [98].
- Its ability to handle dynamic reconfiguration for such types of networks, *i.e.*, WSNs. Namely, a formalism that can show, *e.g.*, links and nodes failure.
- Its flexibility to extend existing models to allow for an early adaptation of formal modelling and verification. That is, more advantages are obtained when applying formal modelling earlier in protocol design while specifying protocols.
- Its ability to support multiple attempts to discover the effects of protocol changes without complex reimplementation to facilitate decision-making.
- Its ability to quickly experiment with different initial states, *e.g.*, by drawing the sensor network topology so that it is possible to generalise the verification results.
- Its simplicity in terms of being understood by a non-specialist audience, including protocol designers who may be unfamiliar with the complex notation often found in formal modelling tools.
- Its extendability to provide extra support if needed, *e.g.*, timing concepts.

Formalisms, such as Petri nets, represent states and transitions in an abstract way. Hence, they are limited to modelling spatial concepts and highly dynamic systems. Event-B, for example, is more focused on event-driven system behaviour rather than structural changes, which makes modelling dynamic changes a challenging task. π calculus is another formalism that can model dynamic connectivity, but it does not support spatial modelling.

Due to the close relationship between graph models and graph-based networks, graph-based formalisms, such as bigraphical reactive systems (BRSs) and graph transformation systems (GTSs), support modelling of WSNs [10]. However, GTSs lack the ability to manage universal quantified predicates, which results in a complicated verification of the properties across all the entities in the model. Additionally, complex rule application orders in GTSs complicate inductive reasoning. Unlike bigraphs, which benefit from their foundation in process calculi, GTSs do not explicitly resolve a concurrent scenario [10].

3.7 Summary

As illustrated, WSN modelling and verification are challenging tasks due to their nature, *e.g.*, dynamic configurations and failures. Although the simulation approach has been shown to be effective, some limitations are defined. Mainly, it lacks coverage of all the possible scenarios. In critical systems, comprehensive verification is required. This chapter presents formal methods (Section 3.1) and formal verification approaches(Section 3.2). It discusses state space explosion as one of the formal verification drawbacks, and it also covers some mitigation strategies to alleviate this issue in Section 3.3. Then, a state-of-the-art method is presented in two sections: Section 3.4 compares formal verification and simulation approaches, while Section 3.5 shows some of the existing works that apply formal verification to WSNs. The discussion section (Section 3.6) illustrates the characteristics that are desired in a formal technique to formally verify routing protocols.

Due to the popularity of RPL, its common characteristics with other protocols and the benefits of formal verification, this thesis investigates the use of bigraphical reactive systems (Chapter 4) to provide a novel BRS model for RPL, *i.e.*, a WSN routing protocol. Importantly, employing bigraphs, as a diagrammatic notion, to represent RPL aspects facilitates the adaptation and generalisation of the bigraphs model across various routing protocols, thereby paving the way for formal modelling in protocol design.

The next chapter presents the foundational concepts that are used throughout this thesis. Then, it extensively covers bigraphical reactive systems (BRSs), a formalism we use to formally model and analyse a WSN routing protocol.

Chapter 4

Bigraphical Reactive Systems

This chapter briefly presents preliminary concepts and definitions that are mentioned throughout the next chapters (Section 4.1). Then, the bigraphical reactive systems (BRSs) are thoroughly covered as a core element of this thesis (Sections 4.2 and 4.3). This chapter demonstrates through a simple example, *i.e.*, a printing system(Section 4.3.1), how the BRSs capture the structural and dynamic aspects of systems. Sections 4.4 and 4.5 provides some of the features and extensions of the bigraphs by expanding the same example, which elaborates on the flexibility of the bigraphs. Section 4.6 illustrates the bigraphER toolkit, the bigraphs implementation tool that this thesis uses, followed by an overview of analysing bigraphs models using model checking. A brief review of the systems that have been modelled by BRSs is given in Section 4.7. This chapter also comments on the use of BRSs for system modelling in Section 4.8 and concludes in Section 4.9.

4.1 Preliminary Concepts and Definitions

This section provides a brief background on the basic theories and models mentioned in this thesis (Chapters 5 and 7).

4.1.1 Markov Chains

A Markov chain is a mathematical system that models the random transition of a system from one state to another within a countable set of states. The transition from one state to the next depends only on the current state and not on the sequence of events that preceded it.

Discrete-Time Markov Chains are a variant of Markov Chains that label each state transition with a probability. Each transition has a probability p, where $0 \leq p \leq 1$, and the sum of all the outgoing edges from a state is equal to 1. The system transitions from one state to another depending on a probabilistic choice that occurs in discrete time steps.

Definition 4.1 (discrete time Markov chains (DTMC))**.** A labelled discrete tim Markov chain is a tuple (S, si, P, AP, L), where S is a set of states, $s_i \in$ S is the initial states (often s_0), P is a transition

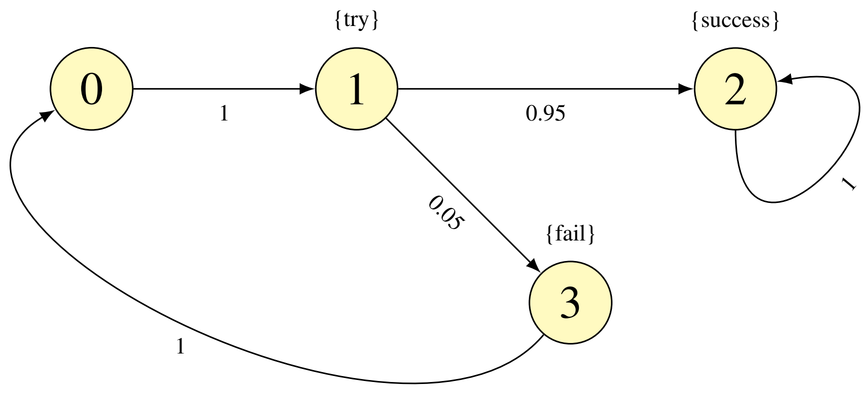


Figure 4.1: An example of a discrete time Markov chain (DTMC).

probability matrix $(P : SxS) \rightarrow [0,1]$, AP is a set of atomic propositions and L $(L : S \rightarrow 2^{AP})$ is a labelling function that maps each state to a subset of the atomic propositions AP.

For each $p_{s,s'} \in P$, $p_{s,s'}$ defines the probability of the next state s$'$ depending on the current state s, where $P(s,s') \geq 0$, with respect to the fact that the sum of all the outgoing edges is equal to 1 for all $s \in S$. If there are no interesting properties to check in a state, the labelling function is often omitted from the tuple.

Example 4.1. Figure 4.1 shows a digram of a DTMC with $S = \{0,1,2,3\}$ and the initial state $S_{init} = 0$. The following is the corresponding probability matrix:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0.95 & 0.05 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Here, $\mathbf{P}(s,s')$ represents the probability of moving from state s to state s'.

4.1.2 Markov Decision Processes

A DTMC models the system that transits from one state to another depending on a predetermined probability for each state. However, when this transition is nondeterministic, Markov decision processes (MDPs) are used to model the outcomes, which are partly random and partly under the control of a decision maker.

MDPs [15, 58] extend DTMCs by permitting a choice of possible actions at each state. That is, a DTMC offers a single probability distribution per state, whereas an MDP permits multiple probability distributions through allowing the choice of action per state. Therefore, a DTMC can be seen as a special case of an MDP where there is only one action available at each state.

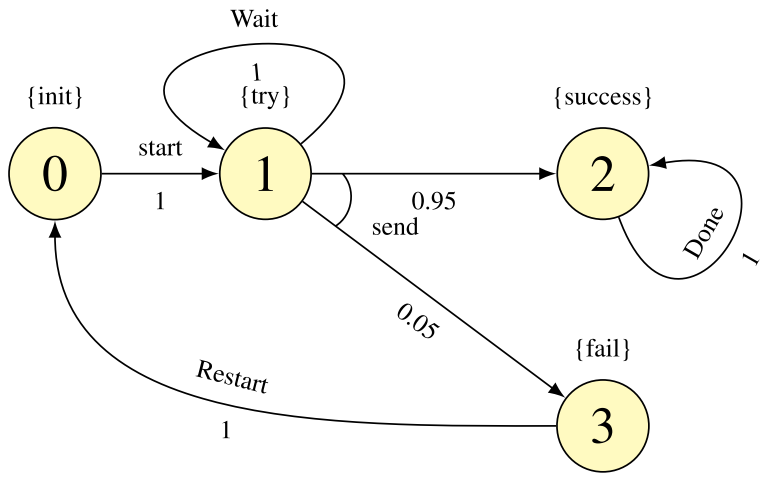


Figure 4.2: An example of a Markov decision process (MDP).

Definition 4.2. An MDP is a tuple $(S, s_i, A, Step)$, where S is a set of states with a predefined initial state $s_i \in S$, often s_0. In each state, there is a choice between multiple actions drawn from set A, and associated probability distributions, as specified by the function $Step$. Probabilities are used to determine the likelihood of one transition over the others.

Example 4.2. Figure 4.2 is an example of an MDP with $S = \{0, 1, 2, 3\}$ and initial state $S_{init} = 0$. The following is its probabilistic transition function:

$$\textbf{Steps} = \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 0.95 & 0.05 \\ \hline 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 \\ \hline \end{array}\right]$$

Note that, for clarification, actions are omitted from the matrix, and we add horizontal lines to separate distributions associated with different states.

4.1.3 Probabilistic Timed Automata (PTA)

Probabilistic timed automata (PTA) [6] is a mathematical model that describes systems that show probabilistic, nondeterministic and timed behaviours. PTA is capable of verifying quantitative properties, *e.g.*, the maximum probability of delivering a message within 3 seconds. Costs or rewards can also be involved to specify additional properties, *e.g.*, the minimum messages that can be delivered within 30 seconds. Clocks are variables that require non-negative real numbers that keep track of time. Clocks are used to describe the real-time behaviour of a system. That is, a clock variable C is a set of non-negative real numbers where $C = \{c1, c2, c3, \ldots\}$ and c_i is a real-valued variable representing time. A clock valuation v is a function that assigns a non-negative real number to each clock variable. That is, $v(x_i) \in \mathbb{R} \geq 0$ for each clock $c_i \in C$.

Definition 4.3. A PTA is a tuple (L, l_0, E, A, C, I), where L is a set of locations (*i.e.*, states) with $l_0 \in L$ as a start location. Here, E is a set of edges that represents the possible transitions between locations, while A is a set of (discrete) actions that may occur in the system. In addition, C is a finite set of clocks and I is a set of clock invariants associated with each location.

The invariants could be flexible; they are typically used to force a transition from a location, *e.g.*, an invariant $x \leq 2$ forces a location move at time 2 if we have not already left the location (alongside transition conditions, these usually encode a "move within a maximum of 2 time units" semantics). Transitions are associated with actions and probability distributions, with the addition of *clock constraints* (*e.g.*, $x > 3$) and *clock resets* (*e.g.*, $x := 0$).

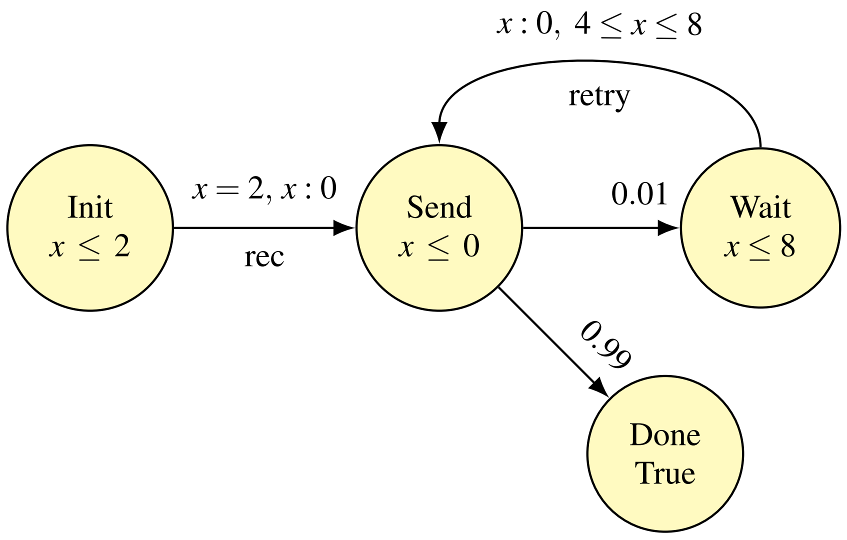
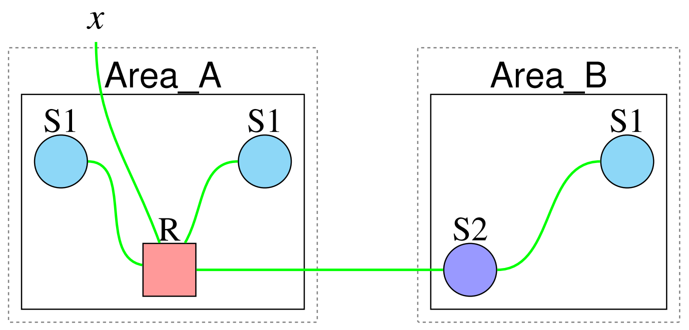
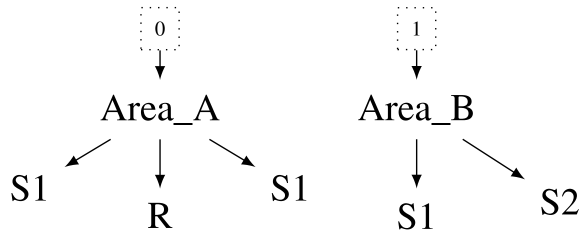


Figure 4.3: An example of probabilistic timed automata for a simple sending data process from [78].

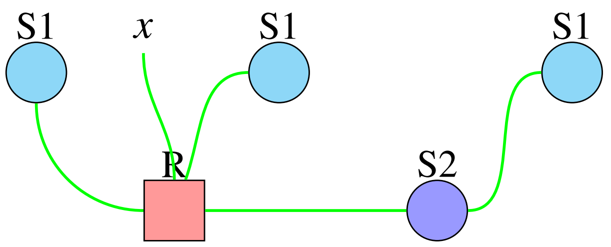
As shown in Figure 4.3, a standard graphical representation for PTAs can demonstrate a simple protocol. The system has one clock x and 4 locations: "Init", "Send", "Wait", and "Done". Starting at location "Init", the system can leave at any time where $x \leq 2$, as specified by the state invariant. However, the system must move to location "Send" once $x = 2$ (specified by the enabling condition $x = 2$ on the action rec), where it resets the clock. Instantly, the system with probability 0.99 moves to the "Done" location, where the system terminates, or to the "Wait" location with probability 0.01. In the "Wait" location, the system waits at least 4 units before retrying to reach the "Send" location. When $x = 8$, the system moves to the "Send" location and resets the clock x so it can send again.



(a)



(b) The place graph.



(c) The link graph.

Figure 4.4: An example of bigraph with two regions: Area_A contains two sensors of type S1 and a Router and Area_B contains one sensor of type S1 and one sensor of type S2.

4.2 Bigraphs

Bigraphs are a universal mathematical modelling formalism introduced by Robin Milner in 2008 to model systems that evolve in both time and space [97]. Bigraphs have been used successfully to model and verify different systems, such as ubiquitous systems [16], cyber-security in smart buildings [145] and biological processes [75].

Bigraphs are a graph-based formalism in which the shapes and colours in the diagram are freely chosen by the modeller. Nevertheless, bigraphs permit equivalent algebraic and diagrammatic notation. Note that this equivalence means that nothing is lost from the theory by using diagrams over algebra, so this change in notation is not a simplification of what a modeller would write. Table 4.1, which is provided below, shows the equivalence between the diagrammatic notation of the components and operations that we used throughout this thesis and their algebraic definition. Henceforth, this thesis will use diagrammatic forms since it allows for a more intuitive description.

As shown in Figure 4.4, a diagrammatic form of bigraphs (Figure 4.4a) consists of a pair of relations for the same set of entities: a place graph (a directed forest, as shown in Figure 4.4b) and a link graph (a hypergraph, as depicted in Figure 4.4c). A bigraph instance, as a combination of a *place graph* and a *link graph*, consists of vertex V and edge E, where the vertices represent the entities of the system being modelled and the edges denote the relationships between these entities, as described below.

4.2.1 Bigraph Constituents

The word bigraphs originates from the fact that bigraphs consist of two graphs: *place graphs* and *link graphs*. In the following, we provide an example-driven description of bigraphs using a simple system.

- **The place graph**

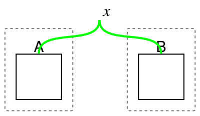
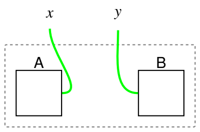
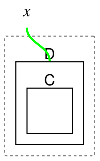
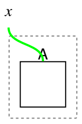
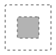
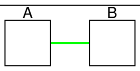
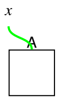
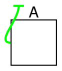
A place graph (Figure 4.4b) models the spatial structure that is contained in a region known as a root. A region is graphically represented by a dashed rectangle to represent adjacent parts of a system. A place graph can contain a site, known as a hole, which represents parts of the system that have been abstracted away. Sites are shown as a filled rectangle in graphical representations that can be empty. Sites can be seen as *variables*, which we can use to specify a bigraph without explicitly mentioning all the elements. Generally, 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). The sites and regions are numbered, beginning with 0, from left to right. They allow for the composition of the placing graphs, as we will see later.

In addition, place graphs contain *entities*, which are user-defined variables that may be related *spatially* through nesting, *e.g.*, a computer in an office, or *non-spatially* via links, *e.g.*, a computer connected wirelessly to a printer. Entities are assigned a type called *control*, which is denoted by a label in the figures *e.g.* S1 and R in Figure 4.4. An entity is *atomic* if it does not contain any other entities or sites. Each entity has fixed *arity* (shown as points on the entities) that determines the number of ports, which denotes the number of links that an entity must have. Note that these ports are unordered.

- **The link graph**

A link graph (Figure 4.4c) is a set of hyperedges that describe non-spatial relationships among entities that represent the interaction between them. Links can be hyperlinks since they may connect 1-to-*n* rather than more traditional 1-to-1 links. Bigraphs permit specifying multiple situations for links to assist in modelling, as shown in Table 4.1. Links can be open, that is, partly specified to indicate that the link may connect one entity to another at a later stage. On the other hand, closed link is fully specified to show a non-spatial relation between two or more entities. Links must always be present, for entities that have 1 or more arity, but might not connect anywhere (a 1-to-0 link), which is shown as an orthogonal line at the end of a link. Links may be named, in which case they are open to extension, *e.g.*, they might connect elsewhere in some larger model (outer names). That is, if it is open, then the link might (or might not) link elsewhere; if closed, then it can *only* link the specified entities. However, we can use a closure operator ($/$) to indicate that the link is closed and no connection is possible. Note that the actual identifier used for links are irrelevant *e.g.*, $/xA_x$ is equivalent to $/yA_y$. Unbound identifier is called a name, an idle name refers to an existing name that is not connected to any other entity or name.

Table 4.1: Equivalent diagrammatic and algebraic definition of various example bigraphs.

Component\Operation	Diagrammatic form	Algebraic form
Parallel product		$\mathsf{A}_x \parallel \mathsf{B}_x$
Merge product		$\mathsf{A}_x \mid \mathsf{B}_y$
Nesting		$\mathsf{D}_x.\mathsf{C}$
Entity of arity 1		A_x
Identity Place Graph		id
Closed link		$/x(A_x\|B_x)$
Open link		A_x
Name closure		$/xA_x$
Idle name	x	{x}

Definition 4.4 (bigraph signature)**.** The set of controls and the number of ports that an entity can have form a *bigraph signature* that takes the form (K, ar), where K is a set whose elements are the controls and *ar* assigns an arity to each control. For example, the bigraph signature for the one shown in Figure 4.4 is $\mathrm{K} = \{Area\_A : 0, Area\_B : 0, S1 : 1, S2 : 2, R : 4\}$.

The composition of the place bigraphs is enabled through regions and sites. That is, a site of one bigraph may be replaced by a region of another. Likewise, outer names permit link graph composition.

Main Points Recap.
Given the example shown in Figure 4.4, we recap the following points:

- Figure 4.4b shows the place graph and Figure 4.4c displays the link graph.
- Vertex = $\{Area\_A, Area\_B, S1, S2, R\}$.
- Edge = $\{x\}$.
- $$Arity(v) = \left\{ \begin{array}{ll} 0 & \text{if v} \in \{\text{Area\_A , Area\_B}\} \\ 1 & \text{if v = S1} \\ 2 & \text{if v = S2} \\ 4 & \text{if v = R} \end{array} \right\}.$$

4.3 Bigraphical Reactive Systems

Bigraphs specify a system at a particular point in time. To demonstrate the system's dynamics, bigraphs are equipped with a set of reaction rules. A reaction rule states that a bigraph evolves by replacing the *matching* bigraph with another. A set of all the bigraphs, closed under the set of rules, is known as *a bigraphical reactive system (BRS)*.

A reaction rule, also known as a rewrite rule, takes the form $L \longrightarrow R$, which means that this rule applies whenever a state (bigraph) matches the state (bigraph) specified in L. The application of the rule substitutes the bigraph L with the bigraph R. In general, a rule can apply whenever

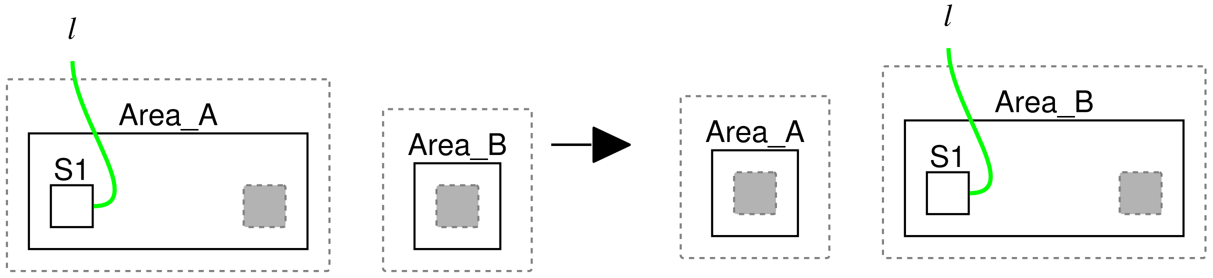


Figure 4.5: Reaction rule `move_Entity`: S1 is moved from Area_A to Area_B

there is an appropriate match. An example reaction rule is given in Fig. 4.5, showing a scenario in which an entity is moving from one area to another. We abstract other entities that we do not care about inside sites. However, sites allow for rewrites to be applied in a wide range of circumstances. This means that this single rule matches the case where Area_A might include other information such as other devices.

Note that, since there are *two* S1 entities in Area_A, the rule applies on both of them even though we only want one of them to move. However, bigraphs offer several extensions and features to gain more control over the rule application. Therefore, they facilitate the controlling of the systems being modelled and rule applications.

We now illustrate these variants and characteristics using a simple example of a printing system. We construct the model as we travel through the next few sections.

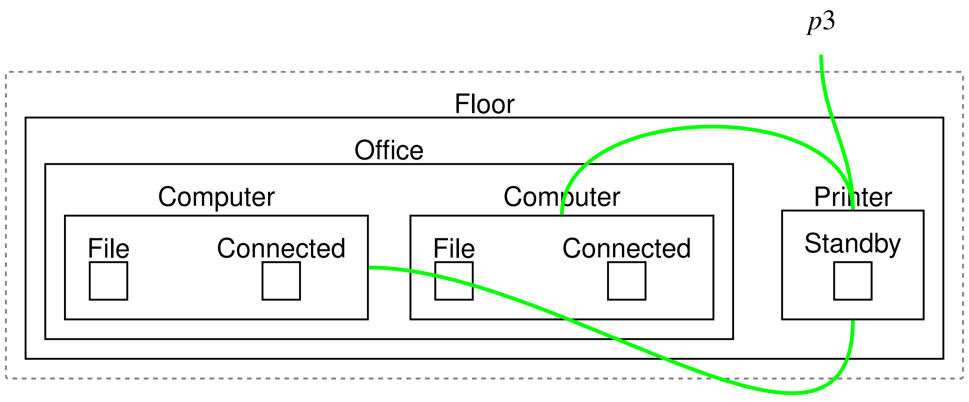


Figure 4.6: An example bigraph for a simple printing system.

4.3.1 Modelling a System Example with Bigraphs

Let us demonstrate BRSs by giving the following example. Suppose that we have a system for a sharing printer across a company. Each floor has one printer; all the computers on a single floor share this printer. Initially, the system is for one floor, but it can be extended to include more floors. The printer handles one file at a time and computers can be connected and disconnected.

To model this system, we start by specifying the structure of the system, as seen in Figure 4.6. A dashed rectangle indicates a region, *e.g.*, a company that contains the following entities: a Floor contains an Office that has *two* Computers and a Printer. Each computer has one arity that enables the connection between the computer and a printer. Each computer also has an entity File, which needs to be printed, and an entity Connected that shows that the computer is connected. In contrast, the Printer has *three* arities, the open one enables connections to more computers if required. The Printer has the status Standby or Printing. Computers send a file when the printer status indicates that it is in Standby mode.

To show how bigraphs evolve, we combine bigraphs with a set of reaction rules specified by the system. For example, the rule `send_File_To_Print` shows that a File is sent from a Computer to the Printer when it has the status Standby (Figure 4.7).

A new computer may also be added and connected to the printer via the open link $p3$, as

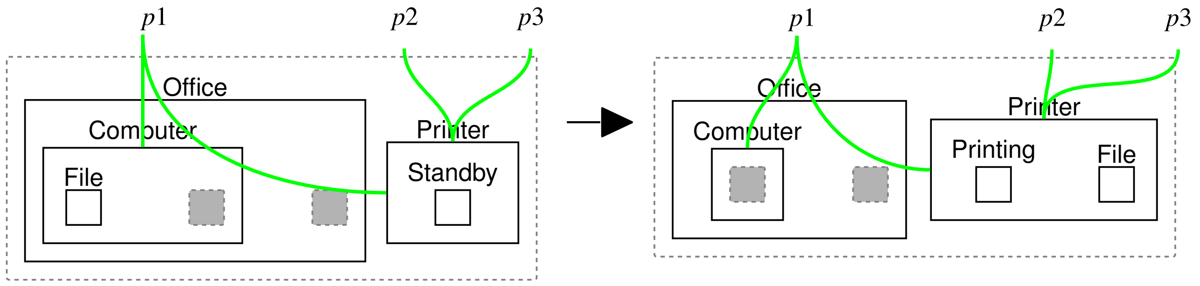


Figure 4.7: Reaction rule `send_File_To_Print`: a File is sent to the printer to be printed.

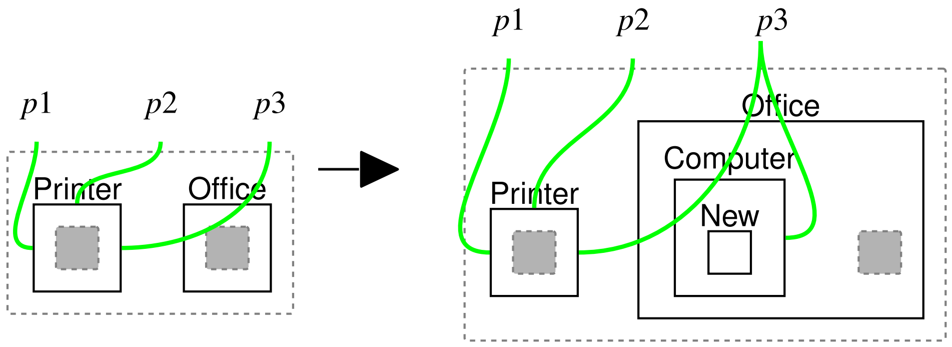


Figure 4.8: Reaction rule `add_Computer`: a Computer is added to the printing system.

shown in Figure 4.8. Another example is shown in Figure 4.9, which disconnects a Computer from the Printer. These *two* rules can be applied an infinite number of times because there is a match each time, adding a computer and then disconnecting it. We demonstrate how this can be prevented later in this chapter. More rules can be encoded, *e.g.*, a rule that returns the Printer to its Standby mode when printing is complete (Figure 4.10). Similarly, we can model more requirements as needed when modifying the system specification, *e.g.*, adding more printers.

4.4 Bigraph Features

Bigraphs have various advanced features that not only help when modelling complicated systems but also facilitate the control of rule applications. The following sections demonstrate bigraph

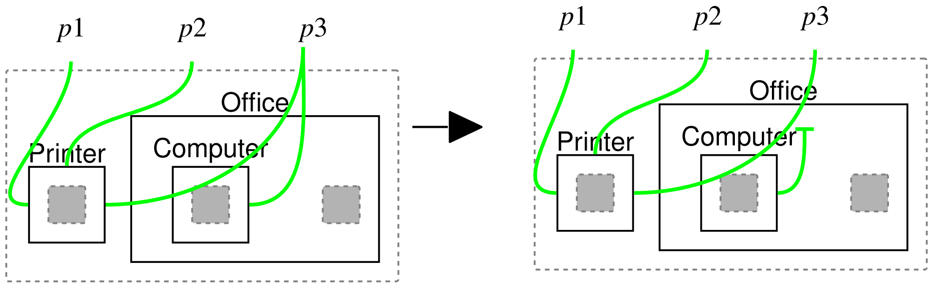


Figure 4.9: Reaction rule `disconnect_Computer`: a Computer is disconnected from the printing system.

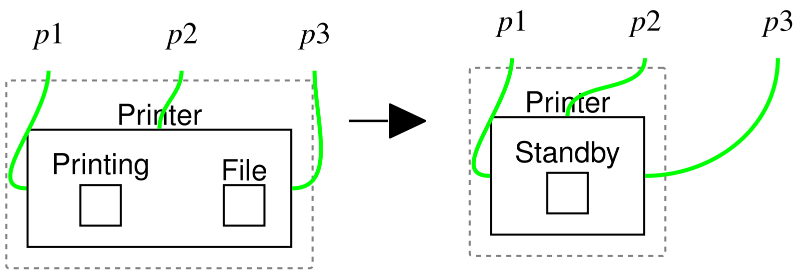


Figure 4.10: Reaction rule `reset_Printer`: a Printer is reset to Standby mode to receive a another file.

features that we will use in later chapters. We provide an informal description, which shows their practical applicability, by adding to the example presented above.

4.4.1 Instantiation Map

Instantiation maps assist when manipulating the content of sites during the application of reaction rules, allowing parts of a bigraph to be duplicated or discarded. An example instantiation map is in Fig. 4.11, which is shown as numbers inside sites. The sites on the right-hand side of the rule copy the bigraph from the similarly numbered site on the left. For example, `copy_Content` rule copies the content of the site 0, *e.g.*, the File from the Computer on the left to the Computer on the right. If there is no site on the right that corresponds to a site on the left, then the contents of the left are discarded. For clarification, we also use dashed blue links that demonstrate which sites are to be copied to which.

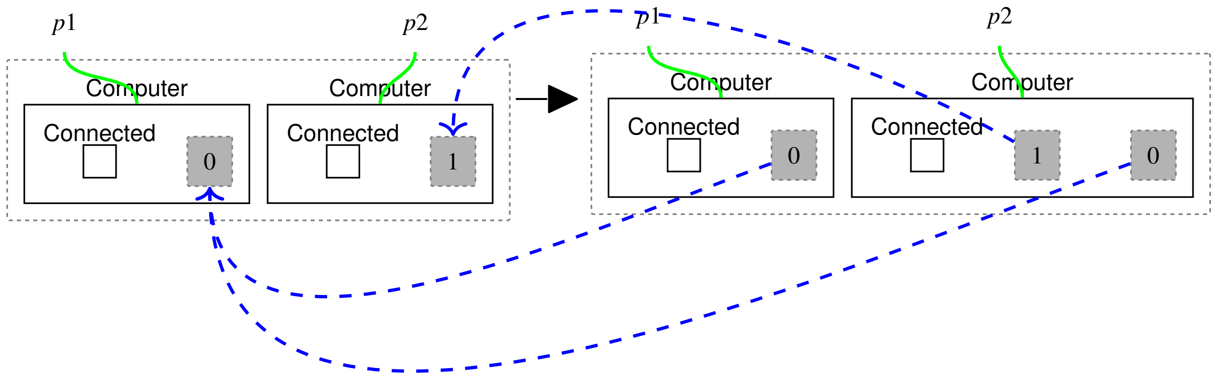


Figure 4.11: Reaction rule `copy_Contents`: copy through instantiation map.

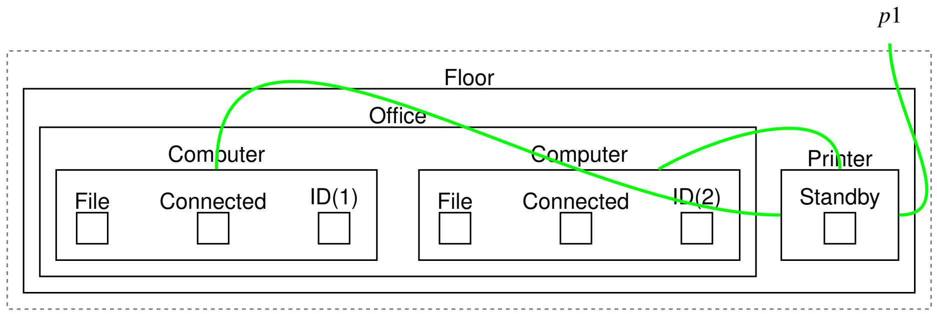


Figure 4.12: An example bigraph for a simple printing system with computer IDs.

4.4.2 Paramaterised Controls and Rules

Entities and rules can be *parameterised* over a set of pre-defined values, *e.g.*, integer, string, etc. For example, the entity IDn, $n \in N$, can be introduced to represent an identifier for each computer in the system (Figure 4.12). This helps to distinguish a specific Computer, *e.g.*, to disconnect it from the system. This is syntactic sugar for ID1, ID2, ...[1]. In addition, numeric operations can also be applied if needed, *e.g.*, n, n+1, etc. Rules instantiated by a parameter assign that parameter value to entities within the rule.

4.4.3 Priority Classes

Basically, a rewriting rule is applicable whenever there is a match. Nevertheless, in some cases, we require that some rules should be applied over others. Therefore, *rule priorities* has been proposed to gain more control over rule applications. That is, for the two rule sets $\{\mathtt{r}_1, \mathtt{r}_2\} < \{\mathtt{r}_3\}$, none of $\mathtt{r}_1, \mathtt{r}_2$ apply unless there is no match for $\mathtt{r}_3$.

Regarding our printing system, we might assign `reset_Printer` with the highest priority since only a single file is meant to be printed at any specific time. This is to guarantee that all the files are smoothly printed.

Instantaneous rules

Instantaneous rules are a special class that consists of silent rules that are applied whenever there is a match but do not appear in the output transition system. It is especially needed for rules that play an important role, but we prefer not to show them in the transition system, *e.g.*, arithmetic operations. Although this type of class is not shown in the transition system, it shares priority with other rules according to its order.

[1]The theory of bigraphs supports infinite entity sets if required, but for practical models the sets are finite.

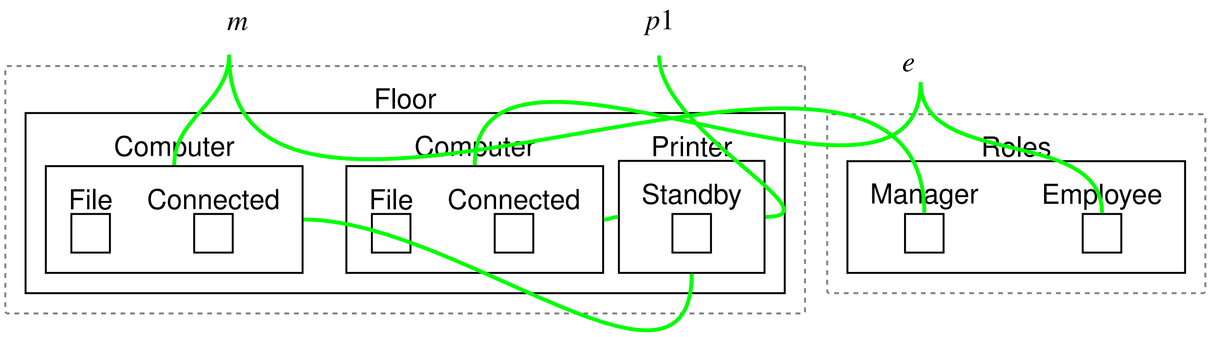


Figure 4.13: An example bigraph for a simple printing system with two perspectives.

4.4.4 Perspective Modelling

Since bigraphs allow composition, this feature can be employed to explicitly separate system components that have different concerns. Given that links can traverse regions, these region (perspectives) can be connected to each others. Additionally, multiple perspectives can be specified at a single reaction rule, identical to rules that are applied on one region only but with the use of the || operator. As an example, we might want to distinguish different computers with different roles (Roles) to restrict access to the Printer. This can be achieved by adding nesting controls, *e.g.*, Manager, Employee, etc. However, to avoid confusing structural and role views, multi-perspective is used, as shown in Figure 4.13. Note that the arity is updated to 2 for the Computer instead of what was initially specified, *i.e.*, 1. The perspective modelling feature provides us with considerable value in our core models, as we will see in later chapters.

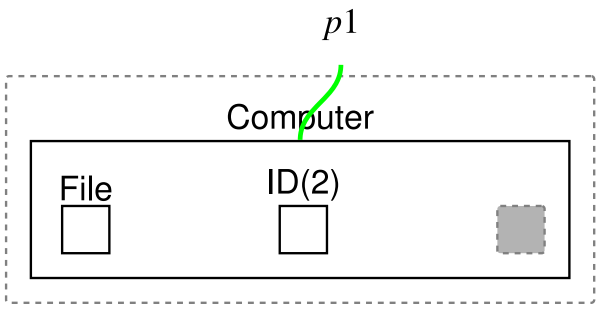


Figure 4.14: A state in which Computer_2 has a file to print.

4.4.5 Bigraph Patterns

To make it easier to write properties for model checking, states can be labelled using bigraph patterns [16] that can label a state whenever a specific bigraph occurs in it. That is, bigraph patterns are static named bigraphs that show a system at a given time. For example, bigraph patterns can be used to mark the state of Computer_2 has a file to print (Figure 4.14).

4.5 Bigraph Extensions

The standard theory of bigraphs has been extended in several works. For example, stochastic bigraphs [75] assign rates to the rules, bigraphs with sharing [128] allow intersecting locations, directed bigraphs [49] associate directions to links, conditional bigraphs [7] add conditions to rules, and probabilistic bigraphs and action bigraphs [9] are extensions to support non-determinism and probabilistic behaviour. More details on the extensions that this thesis uses are given in the following sub-sections.

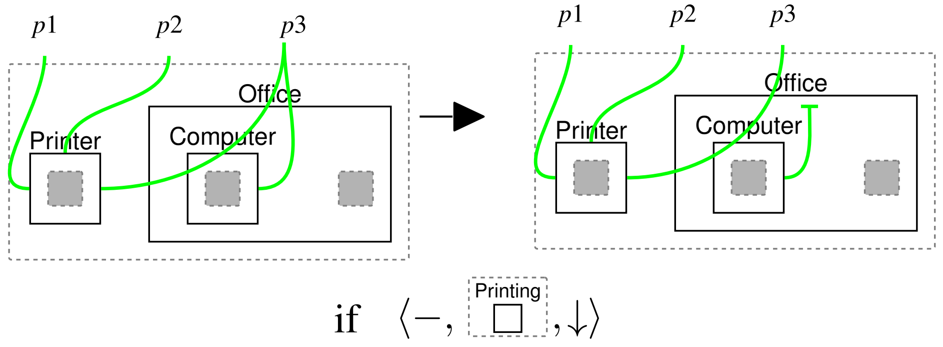


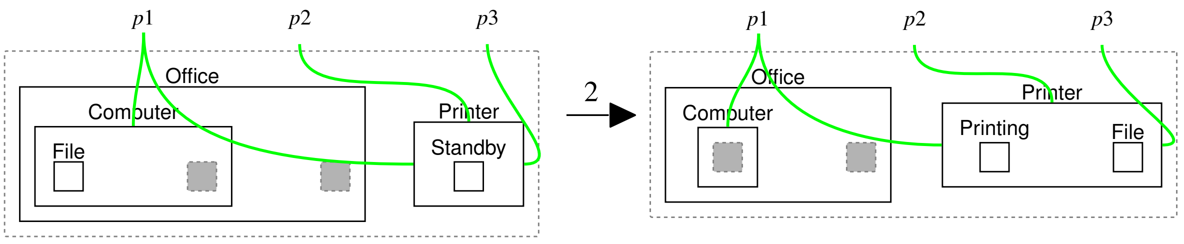
Figure 4.15: Conditional reaction rule `disconnect_Computer`: a Computer is disconnected from the printing system if the printer is not in Printing mode.

4.5.1 Conditional Bigraphs

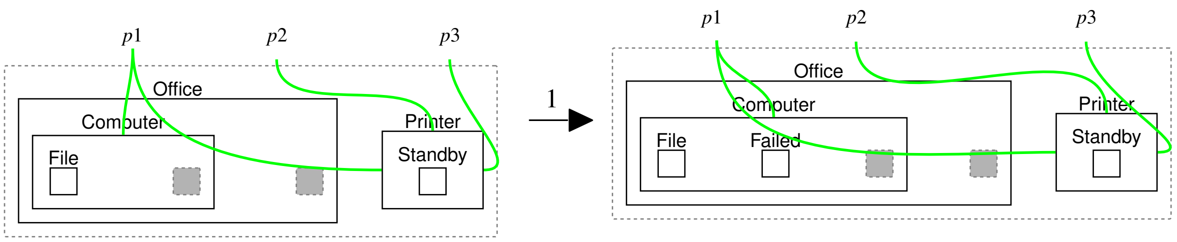
Conditional bigraphs [7] is an extended formalism that extends the standard bigraphs to restrict the applicability of a rewrite rule. That is, a rule can be applied not only when there is a match but when a specified requirement is satisfied within the rewrite system. In a standard rule, bigraph B' replaces bigraph B regardless of the presence or absence of an unspecified entity, which could even be abstracted in a site. A condition specified in a conditional rule is a constraint that must be satisfied for B' to replace B when there is a match. The condition is checked for a rule to be applied on both the context – above the match – or parameters, below the match.

The condition is specified in a reaction rule as $\text{if} \;\; \langle + \setminus -, \overset{\text{bigraph}}{\square}, \downarrow \setminus \uparrow \rangle$ where $+ \setminus -$ represents the presence ($+$) or absence ($-$) of the entity (*e.g.*, bigraph). The downwards arrow indicates checking for the presence or absence of an entity in the parameters (*i.e.* sites and the left-hand side) whereas the upwards arrow denotes checking everything else in the model. The importance of conditional bigraphs is obvious with the existence of the sites. For example, several rules in our printing examples apply for multiple (or even infinite) times as the match is always found. For example, in Figure 4.9, the rule for disconnecting a Computer is to be applied for all connected computers but we add a condition that only if the Printer is not in printing mode to restrict the application. This can be specified by adding the condition that $\text{if } \langle -, \overset{\text{Printing}}{\square}, \downarrow \rangle$, as shown in Figure 4.15.

Note that both priorities and conditions allow us to control the application of the rules; however, it is preferable to use conditional bigraphs instead of priorities. This is because the conditions reveal the purpose of a single rule, whereas priorities specify relationships across the whole set of rules.



(a) Probabilistic rule: `send_File_Success`.



(b) Probabilistic rule:`send_File_Fail`.

Figure 4.16: Probabilistic reaction rules.

4.5.2 Probabilistic Bigraphical Reactive Systems (PBRSs)

A probabilistic bigraphs [9] is a variant of the standard bigraphs, where the probability of transiting from a given state (bigraph) to the next one is controlled. That is, the states (bigraphs) resulting from reaction rule applications are generated from a probability distribution. Each reaction rule is assigned a weight, which is normalised to a probability that depends on which reaction rules find a match. In other words, among multiple applicable rules, the probability of a specific rule application is given by the weights that specify how likely this rule will apply with respect to all the other applicable rules.

The probabilistic reaction rule takes the form of $L \xrightarrow{p} R$, where p is the weight. For example, we extend the `send_File_To_Print` rule that sends a File to be printed by including the probability that the File sends successfully to be *twice* as likely as it failing (Figure 4.16).

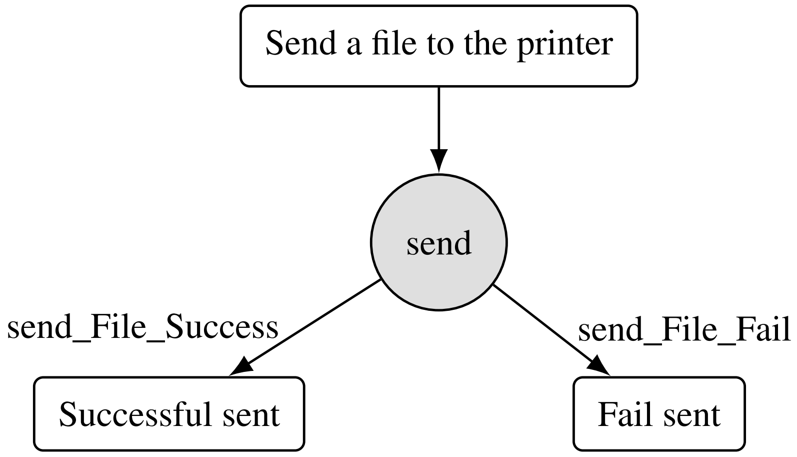


Figure 4.17: Action bigraphs transition system.

4.5.3 Action Bigraphical Reactive Systems (ABRSs)

Probabilistic bigraphical reactive systems are extended to enable the modelling of an explicit non-deterministic action choice. An action is a non-empty set of weighted rules, where their applicability is affected by the action they are in. That is, when an action is selected, only the reaction rules that are in this action can be applied. Note that these rules are applied as specified in probabilistic bigraphs. An action is applicable whenever there is one reaction rule (or more) from this action is applicable. Rewards can also be associated with actions and the underlying MDPs can be verified using off-the-shelf model checkers, such as PRISM [77].

As an example of ABRSs, we can use the reaction rules in Figure 4.16 and map explicit *actions* to two of the reaction rules. That is, we enclose the *two* rules with the *send* action (*i.e.*, $send = \{send\_File\_Success, send\_File\_Fail\}$) so that at least one of the two *send* rules is applicable when the action *send* is chosen.

4.5.4 Other Extensions

It is noteworthy that there exist other extended formalisms for bigraphs, which we decided not to use despite their suitability for the models that are demonstrated in this thesis. Bigraphs with sharing and BigActors may replace what we have done for our models, but we decided not to employ them for the following reasons:

1. **Bigraph with sharing** [128] is a bigraph extension that converts place graphs into directed acyclic graphs, instead of a forest. This results in entities maybe having any number of parents, which helps in, *e.g.*, the modelling of overlapping or intersecting locations. Our approach, *i.e.*, the models we employ later, is equivalent to what could be done with a bigraph with sharing. However, it allows us to easily type links when modelling broadcasting with link status indicators, *e.g.*, the idle status.

2. **BigActors** [111] is a formalism that is used to separately model the physical and logical

structures of the system. That is, the physical and virtual layers are modelled using bigraphs and agents, respectively. In our model, which is shown later, we use bigraphs in its original version but we use bigraph perspectives.

It should be noted that bigraphs mean conditional bigraphs in Chapter 5 and action bigraphs in Chapter 7.

4.6 Bigraph Implementations and Verification

There exist multiple bigraph implementation tools in the literature that assist in modelling systems and applications in the form of BRSs. JLibBig [28], for example, supports directed bigraphs but does not support probabilistic rewriting or conditionals. Other implementations such as Bigraphical programming language (BPL) [56] and bigraph model checker (BigMC) [113], are currently unmaintained. LMNtal [146] is a programming and modelling language for general hierarchical graph rewriting. It supports bigraphs, as well as other formalisms such as (coloured) Petri nets and π Calculus. A recent tool, Bigraph Toolkit Suite[2], is a Java-based framework implemented to create and simulate bigraphs with a basic model checker included. In this thesis, we utilise BigraphER toolkit [127] to provide implementations for all the covered works and examples due to its features and characteristics, as illustrated below.

4.6.1 BigraphER

BigraphER is an open-source command-line toolkit used to manipulate bigraphs and their constituents and reaction rules through OCaml library programming interfaces. It simulates bigraphical reactive systems, and also computes their transition systems via algebraic representation of the initial state and reaction rules. BigraphER supports rules with instantiation maps, rules priorities and paramartised controls and rules. It allows the state to be labelled and, hence, be formally checked. BigraphER generates a graphical output of bigraphs and also supports several transition system formats. Therefore, it allows for the the use of model checking tools, *e.g.*, PRISM. BigraphER supports different bigraphs extensions, such as action bigraphs, probabilistic bigraphs and bigraphs with sharing. The BigraphER webpage provides a variety of bigraph model examples. Furthermore, BigraphER is well-maintained and consistently updated according to recent extensions. Regarding its syntax, it possesses a syntax similar to the algebraic definition of bigraphs.

We now provide a concise overview of the BigraphER format by giving a single example for each part of the bigraphs implementation, which is specified in BigraphER. A comprehensive guide on the use of the BigraphER tool in specifying bigraphs models is provided in [8]. To model a system using BigraphER, one should specify its *five* main parts as follows.

[2]https://bigraphs.org/index.html

1. Controls definitions that specify the arity and whether it is atomic and\or parameterised controls:

```
ctrl Office  = 0;
ctrl Floor = 0;
ctrl Computer = 1;
ctrl Printer = 3;
atomic ctrl File = 0 ;
atomic ctrl Standby = 0 ;
atomic ctrl Connected = 0;
atomic fun ctrl ID(i) = 0;
```

2. Reaction rule, which specify the rewriting step by defining the matching bigraph and the replacement one:

```
react send_File_To_Print =
         Office.(Computer{p1}.(File | id ) | id )
       | Printer{p1,p2,p3}.(Standby )
  -->
          Office.(Computer{p1}.(id ) | id )
        | Printer{p1,p2,p3}.(Printing | File  ) ;
```

3. Predicates that assist in the labelling of interesting bigraphs (this is optional):

```
#Predicate

big multiple_receive = Printer{p1,p2,p3}.(Printing | File | File );
```

4. An initial state definition:

```
#Initial State

big office =
     /p1/p2 (
             Floor.(
              Office.
                ( Computer{p1}.(File | Connected | ID(1))
                | Computer{p2}.(File | Connected | ID(2))
                )
                  | Printer{p1,p2,p3}.Standby
                   )
            );
```

5. A BRS definition block:

```
begin brs
int i = {1,2};
init office;
   rules = [
              {send_File_To_Print}
            ];

  preds = {multiple_receive};

end
```

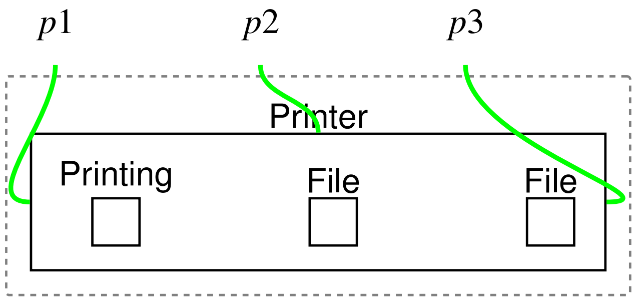


Figure 4.18: A state with “multiple_receive”: a printer receives two files at the same time. This should never happen.

4.6.2 Model Checking for Bigraph Models

Bigraph models can be formally verified by existing model-checking tools such as PRISM [77], allowing the properties of temporal logic to be checked. BigraphER generates the complete transition system that can be used as input for model checkers. To make it easier to write properties, we utilise bigraph patterns(Section 4.4.5) so that we can label an interesting state whenever a specific bigraph occurs in it.

Using the LTL mentioned in Chapter 3, as an example, we specify that a Printer should not receive a new File if it has the status Printing. That is, we label a state as “multiple_receive”, which shows if there exists *two* files nested in the Printer (Figure 4.18). Namely, we label a system state in which there are *two* files as “multiple_receive”. We specify the following properties to check that this should never happen:

$$\mathbf{A}\,[\,\mathbf{G}\,\neg\texttt{multiple\_receive}\,]\,. \tag{4.1}$$

4.7 Modelling Applications with Bigraphs

Bigraphs have been successfully used in modelling and verifying different systems such as ubiquitous systems [16], cyber-security in smart buildings [145], cloud systems [123], sensor network infrastructure [129], rational agents [11] and biological processes [75]. The following presents some examples of works that use bigraphs in system and network modelling.

Wireless Mesh Networks. Wireless mesh network topology is composed of stationary wireless access points mesh routers (MR) and mobile client nodes. MR is typically equipped with several network interface cards (NICs) that operate on either the same wireless access technologies or different ones to improve the flexibility of WMNs [22]. Wireless mesh routers can communicate simultaneously through multiple channels. A Wireless Mesh Network (WMN) consequently may become a multi-channel, multi-interface network. Simultaneous transmissions on the same channel can lead to collisions and, hence degraded performance. Boucebsi and Belala [22] have merged the logical reflection of Maude language and the hierarchical structure of the bigraphical reactive systems to offer an implementation formal model for wireless mesh networks in order to address the interface problem, which considerably reduces WMN performance.

System of Systems. Bigraphs are also used in modelling system of systems, which are composed of large-scale integrated systems that are networked together for a common goal but independently operate on their own. Bigraph can model both structural and behavioural characteristics. The former can be reflected by the bigraphical concepts of linking and locality. The latter requires further capabilities, besides linking and locality concepts, to reflect the dynamic progression of the system. Reaction rules enable the description of the change; however, the process of implementing the dynamics of bigraphs faces the matching problem. The process of bigraph matching involves computing whether and in what way a reaction rule R can be applied to replace the given bigraph B [43]. Milner et al. [34], provides the first work that addresses bigraph matching, that proposed a matching system over bigraph terms for inferring legal matches of bigraphical reactive systems and, the work results in implementation of the BPL. The proposed matching system occurs after the implementation of various steps such as normalisation, renaming and regulation; thus, the computational time could be increased. However, ref. [43] formalises BRSs as graph transformation systems (GTSs), instead of implementing a matching of bigraphs, which benefits from the existing tools developed for graph transformations since a BRS is closely related to graph transformation. Thus, this work presents a bigraph rewriting tool, which is called BiGMTE (bigraph matching and transformation engine), that is based on a graph matching and transformation tool named GMTE. The tool simulates the behaviour of bigraph transformations by simulating their translation using the transformation tools and techniques of the graph.

Self-Adaptive Fog Systems. Bigraphical reactive systems have been used to model self-adaptive fog systems. The goal of Fog computing is to to decentralise cloud systems that distribute microdata centres in the core network and resource-scare devices at the edge of the network [124]. Fog systems involve exploiting resources around the edge to proceed latency and delay-sensitive tasks adjacent to the end users resulting in avoiding network bottlenecks. The architecture of the fog system is a set of diverse computing nodes dispersed across separate layers, the lower one is the edge layers. Given that a BRS can represent the hierarchy of the entities and applications of fog-based systems, alongside their characteristics and the complex interactions between the several elements, it is used as a formal foundation for the modelling of fog-based systems. Moreover, the reaction rule concept is well-suited for capturing the temporal evolution of fog systems, and for producing a sequence of bigraphical models that show the past and current states of the fog system. In addition, defining the spatio-temporal properties related to such systems' behaviour and application requirements could be implemented by the bigraph patterns in combination with standard boolean operators or more expressive temporal logic operators. Furthermore, a BRS is described by a human-oriented graphical aspect that equals the representation of the algebraic ones, which provides an intuitive representation of the fog-based systems. For all these features, a BRS is an appropriate approach to structurally and behaviourally model fog systems [124].

Large-Scale Sensor Network Infrastructures. BRSs have been utilised in modelling WSN infrastructures on a large scale to support reasoning about the spatial, operational and behavioural aspects [130]. The study uses RBSs to model a real-life urban environmental monitoring infrastructure and its dynamic behaviour. It uses bigraph patterns to express application properties. Then, it uses BigraphER for automated model evolution and reasoning about bigraph patterns and Cooja for generating events and data streams. The result of the experiment shows that the employed approach was able to handle the online verification of large-scale infrastructures that comprise up to 200 nodes.

4.7.1 Bigraphs and Real-Time Systems

Although bigraphs have been utilised in the modelling of various applications, they have deficiencies when modelling real-time systems. In other words, bigraphs are limited for certain modelling requirements such as (a) "in the case of sending a file to the printer fails, the system should be able to resend the file after, *e.g.*, 3 time units elapsed" or (b) "the printing process should takes 2 time units only". This is because the rules for bigraphs are triggered whenever there is a match, but standard bigraphs and their extensions do not allow for such timed events. However, this thesis will provide a modelling strategy that supports real-time systems modelling (Chapter 7).

4.8 Discussion

With regard to Section 3.6, which stated the desired characteristics in a modelling formalism according to the characteristics of the routing protocols, and the previous description of bigraphs, this thesis investigates the use of BRSs to model routing protocols based on the following. A bigraphs model is open to extension, which offers a great opportunity for protocol designers to test their standards while specifying them. That is, the readiness of standards is not a prerequisite for the formal verification process of protocols. Moreover, the rewriting rules for the bigraphs are user-define. Thus, the protocol designers can specify their own writing rules as required. Bigraphs are capable of handling a dynamic reconfiguration for these types of networks. In addition, the existence of tools such as BigraphER enables protocol designers to build practical models. Furthermore, the output format of such a tool enables them to formally check their models using model checkers, *e.g.*, PRISM.

Standard bigraphs do not support explicit clocks; thus, they might not be able to model the timing aspects that WSN routing protocols usually contain. However, given that bigraphs are open to extensions, clocks can be introduced into bigraph theory. This thesis investigates this idea with the aim of empowering bigraphs with clocks, which we will see in Chapter 7.

4.9 Summary

This chapter provided a detailed exploration of the bigraphical reactive systems (BRSs) formalism through Section 4.2 to Section 4.5. The sections demonstrate their primary and advanced characteristics and also the extensions that this thesis employs in the following chapters. Section 4.6 provides an overview of practical modelling in BigraphER, followed by a demonstration of the use of bigraphs in modelling a variety of systems (Section 4.7). The chapter ends with a discussion on the suitability of bigraphs when modelling routing protocols (Section 4.8).

The following chapter thoroughly demonstrates how bigraphs model a real application for WSN protocols, *i.e.*, a routing protocol for low-power and lossy networks (RPL), a popular routing protocol for such a scenario.

Chapter 5

Routing Protocols Modelling

Given the effectiveness of applying formal verification to systems and protocols and the popularity of RPL, this chapter investigates applying bigraphs to verify routing protocols using RPL as an example. The same approach is expected to be applicable to other similar protocols.

This chapter presents a bigraphical model for the key component of RPL, *i.e.*, a destination-oriented directed acyclic graph (DODAG). It first provides a comprehensive description of the modelling steps in Section 5.1, which consists of the following. The network static topology model is given in Section 5.1.1, while Section 5.1.2 and Section 5.1.3 illustrate the RPL dynamic model that results in modelling all possible valid DODAGs. The ability of the bigraph model to be extended is demonstrated in Section 5.2 through several extensions, such as security attack examples and probabilistic extensions. The chapter then discusses some thoughts relating to the use of bigraphs in modelling routing protocols in Section 5.3 and concludes in Section 5.4.

5.1 RPL Modelling with Bigraphs

This section provides a general extendable RPL model that is applicable to any fully connected network topology consisting of n nodes, where there will always be a path to all the nodes, without self-loops. The model focuses on building valid DODAGs as a first step, but is extended to include other features or aspects as required. Using BigraphER, we encode a formal executable model that generates a transition system formatted appropriately for compatibility with model checkers such as PRISM.

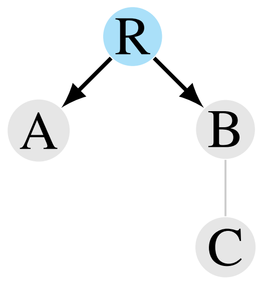
This thesis considers that the *non-storing* mode of RPL, known as RPL-Lite[1], *i.e.*, routing tables, are only maintained at the root and routing information is included in downwards packets. Furthermore, the focus is on the construction of the RPL DODAG assuming a single RPL instance, and using objective function OF0 [142] that assigns uniform weights to links. Henceforth, for simplicity, this thesis uses the abbreviation DAG instead of DODAG.

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

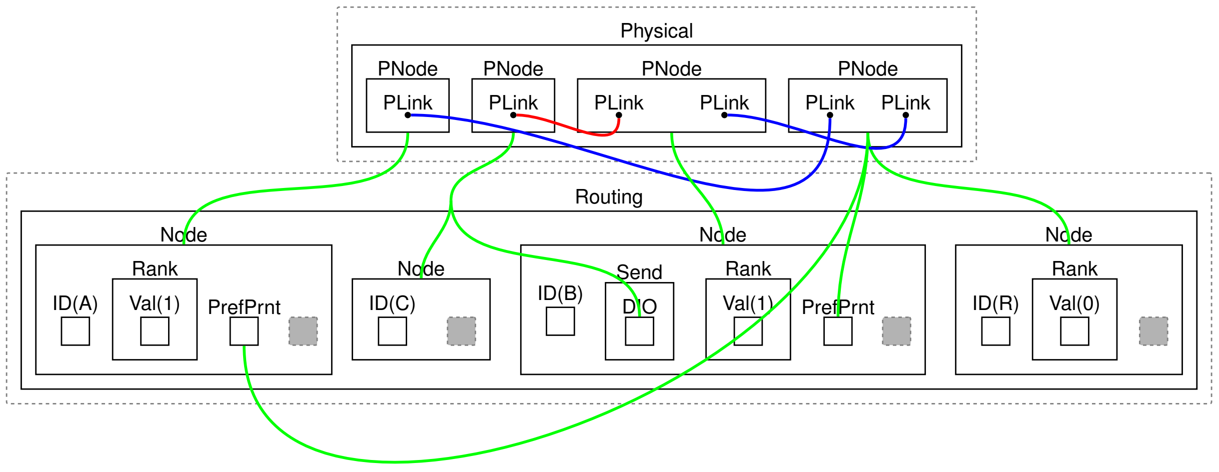
The following work presents a detailed description of how to model the RPL as a BRS. It begins by illustrating how a given topology is modelled and then shows how the RPL constructs a DAG by adding dynamics.

5.1.1 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. The RPL then *overlays* a *routing* topology to provide unicast communication between nodes. That is, although many nodes might be able to communicate physically, the RPL selects which *should* communicate.



(a)



(b)

Figure 5.1: (a) Example WSN topology and (b) corresponding bigraph representation. The partial RPL DAG construction is shown in the physical and routing perspectives.

We use perspectives to capture these two topologies, as shown in Figure 5.1b. 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. Figure 5.1a 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 blue.

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 that the 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 assign 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 5.2).

Routing Perspective

Contains information about all the 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, Nodes A and B link to Node R as their parent. Each Node in the routing perspective is connected to the corresponding 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 modelling.

5.1.2 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 recall the RPL process described in Section 2.3.1 as our starting point to informally illustrate the steps required to construct a DAG as follows. Initially, the root multicasts a DIO to advertise the DAG to any nodes in range. A node that has not already joined the DAG, on receiving the DIO, can join the DAG by selecting the sender as its *preferred parent*. If the node is already part of the DAG, 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 DAG through the same process. Nodes not yet in the DAG may periodically send DIS messages to solicit more DIO

Table 5.1: RPL model entities.

Entity	Arity	Atomic	Description
Physical	0	X	Physical perspective
Routing	0	X	Routing perspective
Node	1	X	Virtual node
PNode	1	X	Physical node
PLink	1	X	Physical link
ID	0	✓	Node's identifier
GenDIO	0	✓	Generation DIO token
DIO	1	X	DIO control message
DAO	2	✓	DAO control message
Idle	0	✓	Link status is Idle
In-use	0	✓	Link status is In-use
Send	0	X	Send messages buffer
Receive	0	X	Receive messages buffer
PrefPrnt	1	✓	The preferred parent for a node
Joined	0	✓	Joined Node
Rank	0	X	Indication of a node has a rank
Value	0	✓	The rank of a node
ClearMulticast	0	✓	Multicasting is done
IncRank	0	X	To increase the received rank by 1
True	0	✓	Indication of new rank is less than the current one
False	0	✓	Indication of new rank is greater than the current one
LT	0	X	Prepare ranks for comparison
L	0	X	To hold a copy of the current rank
R	0	X	To hold a copy of the current rank + 1 (possible rank)

messages if required. Eventually, all the reachable nodes will be part of the DAG and the initial route construction is complete.

We also assume the following:

1. We only model downwards 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 (Chapter 7).

2. We assume nodes receive one DIO at a time, *i.e.*, that the processing time is sufficiently short relative to the message transmission time, and assume no packet drops in the network. Note that, because we undertake full model checking, all the 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 enable efficient verification and ensure it is succinct enough to be described, but they have no effect on the construction of valid DAGs. We can remove these assumptions by adding additional reaction rules, similar to those shown in Section 5.2. The 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, which will be presented in Chapter 7.

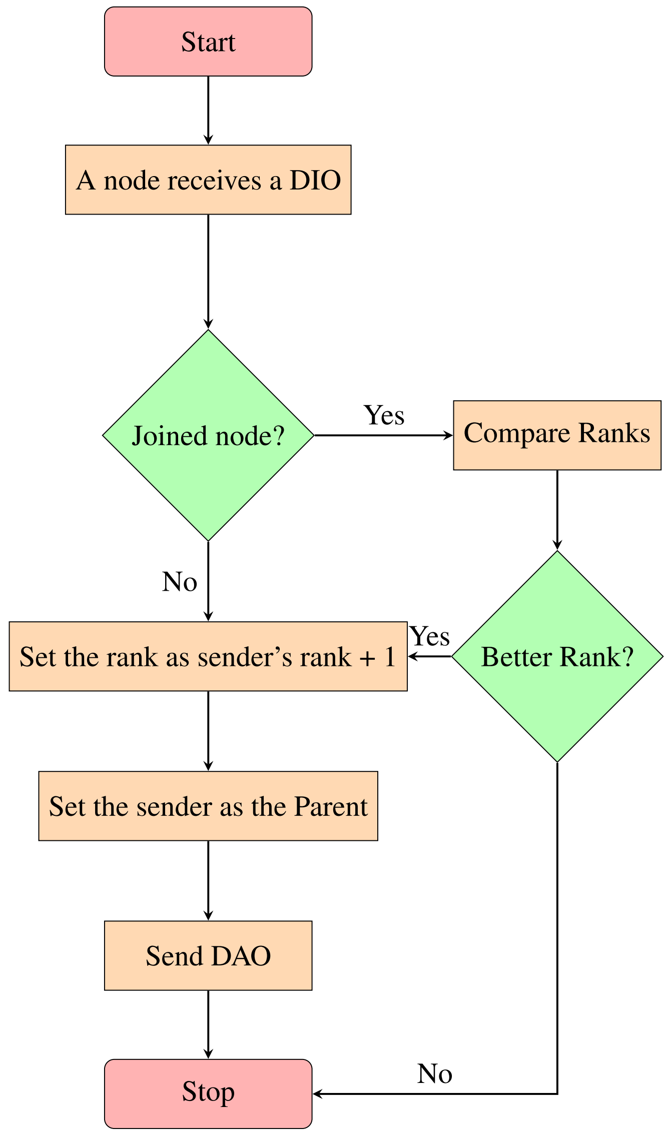


Figure 5.2: Flow chart for the DAG construction based on DIO multicasting

5.1.3 Control Messages Modelling

We use different controls to refer to control messages, *i.e.*, DIO and DAO. Additionally, we use other controls, *i.e.*, Send and Receive, to indicate the direction of the message for each node. A full list of the controls used is given in Table 5.1.

Our model consists of 16 reaction rules (including two parameterised rules) encoding the four main steps of the RPL DAG constructions, as shown in Figure 5.2 for each node in the topology. To ensure the proper execution of the above steps, the priority classes (Section 4.4.3) are employed. Note that rules inside parentheses represent the instantaneous rules that apply repeatedly until no further matches are found, whereas rules inside curly braces indicate standard rules that apply once, after which other higher-priority rules are retried for any new matching occurrences.

```
  rules = [
(handleDIONotJoined, handleDIOJoined, handleDIOLT, handleDIOGT),
{sendDIO},
(incRank(n),lt_l_0, lt_r_0, ranksComparison(n',m')),
(sendDIO_Done, clearMulticast, clearMulticastEnd),
(sendDAOUpParent),
{sendDAO},
(clearDAO),
(generateDIO)
];
```

To keep accurate to the distributed nature of the network, rules rely on updates based on local information only. We provide the diagrammatic versions of the reaction rules here, *i.e.*, the BigraphER implementation, which is similar to algebraic form, including the priority classes are given in Appendix A.

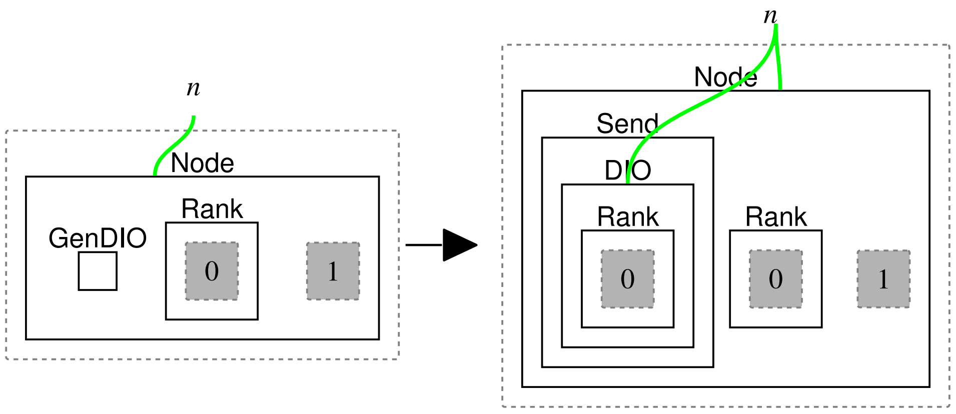
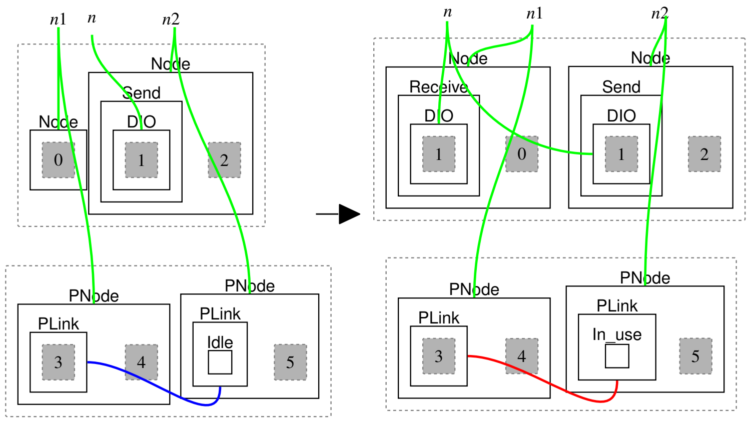


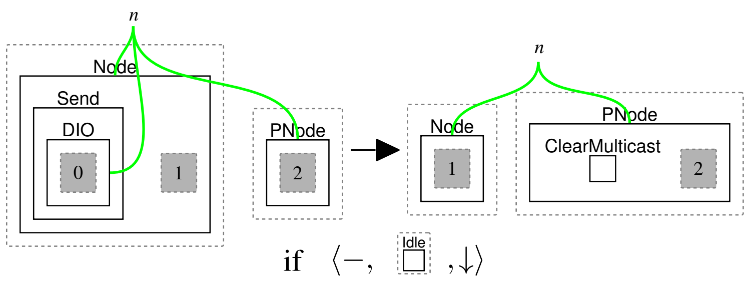
Figure 5.3: Reaction rule `generateDIO`.

Step 1: Generating DIOs (Figure 5.3). 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 instantiation map) of the Rank *at the time the message was sent*[2]. The link to the sending

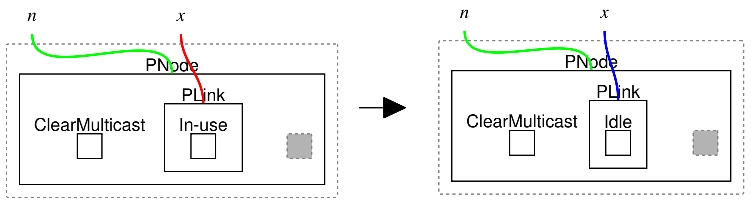
[2]We cannot rely on matching via the link to the sender here, as the rank might update after the message is sent but before it is received.



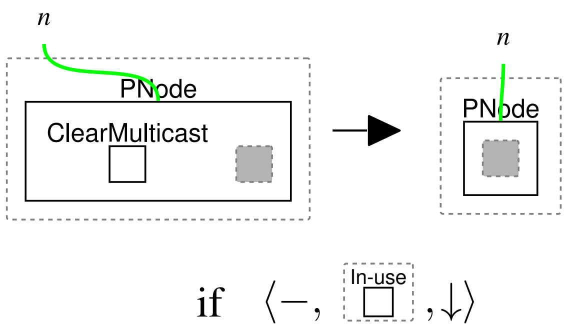
(a) `sendDIO`



(b) `sendDIO_Done`



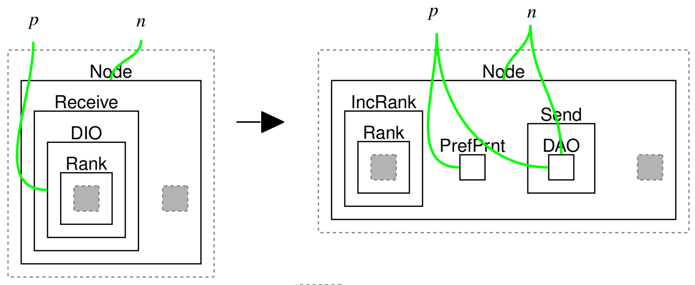
(c) `clearMulticast`



(d) `clearMulticastEnd`

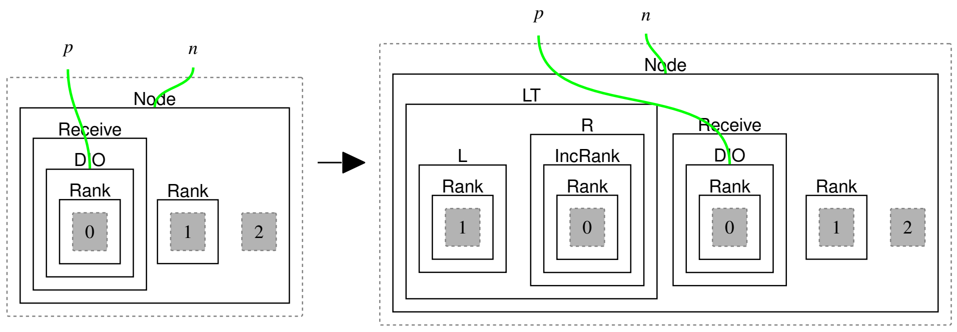
Figure 5.4: Reaction rules for DIO multicasting process.

node allows it to be identified, without needing to explicitly track IP addresses (as would be sent in an implementation).



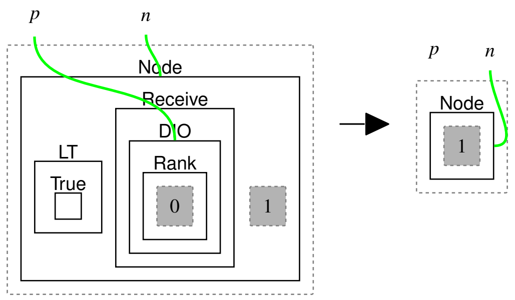
if $\langle -, \boxed{\text{Rank}}, \downarrow \rangle$

(a) `handleDIONotjoined`

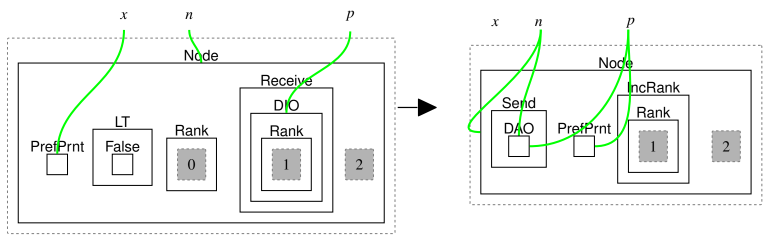


if $\langle -, \boxed{\text{LT}}, \downarrow \rangle$

(b) `handleDIOJoined`



(c) `handleDIOLT`



(d) `handleDIOGT`

Figure 5.5: Reaction rules for handling the DIO.

Step 2: Multicasting DIOs (Figure 5.4) Once we have a message to send, the multicast process can start. Since we want to send a message to all physical neighbours at once, we utilise a multi-step process. First, the rule `sendDIO` (Figure 5.4a) 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 it 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` (Figure 5.4b) removes the Send entity and its content once we detect that all the sends have been completed. This is achieved by specifying a condition checking that there are no Idle physical links. The rule `clearMulticast` (Figure 5.4c) reverts the link used for multicasting into its initial state (Idle) while the rule `clearMulticastEnd` (Figure 5.4d) 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 occur at 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, which leaves message-sending rules to be handled together.

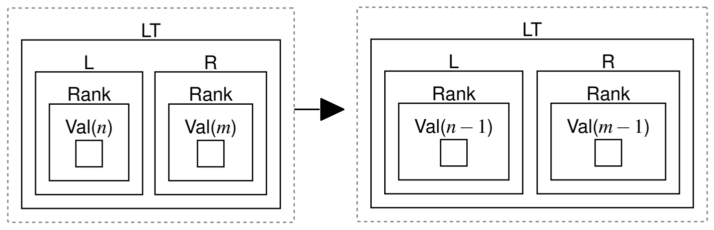
Step 3: Handling the DIO (Figure 5.5). Once a DIO is received, nodes determine how to handle the message. There are three cases for the receiver: (1) not yet joined, (2) joined and the sender has a rank *equal to* or *greater than* the receiver, and (3) joined and the sender has a rank *less than* the receiver.

Rule `handleDIONotJoined` (Figure 5.5a) handles case (1) knowing that 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 $+ 1$[3], 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.

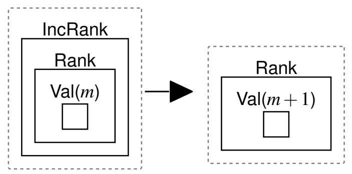
To allow rank comparisons to be generic, and as bigraphs have no built-in mathematical comparisons, we use the rules in Figure 5.6 to perform rank comparisons and increments, where n is the current rank for the receiver and m is the potential new rank. The comparison functions follow the standard recursive definition of less-than via repeated subtraction. We use True and False entities (Figure 5.6c and Figure 5.6d) as flags for the comparison result to be used in the next step; *i.e.*, True indicates that no rank update is required.

The rule `handleDIOJoined` (Figure 5.5b) handles case (2) when the node is already present in the DAG (*i.e.*, it has a rank). Since the sender has a rank equal to or larger than the received rank then the rule `handleDIOLT` (Figure 5.5c) applies and the receiver drops the

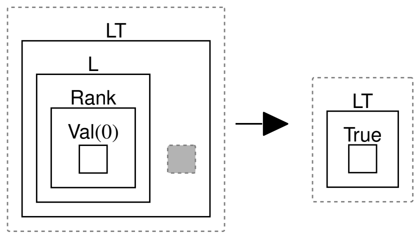
[3]As required by OF0.



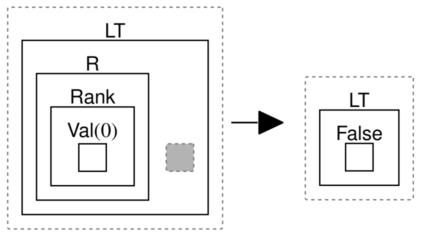
(a) `ranksComparison(n,m)`



(b) `increaseRank(m)`



(c) `lt_l_0`



(d) `lt_r_0`

Figure 5.6: Reaction rules for rank calculation and comparison.

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 a rank less than the receiver (case 3), then `handleDIOGT` (Figure 5.5d) applies to allow the receiver to set a new preferred parent and update to an improved rank.

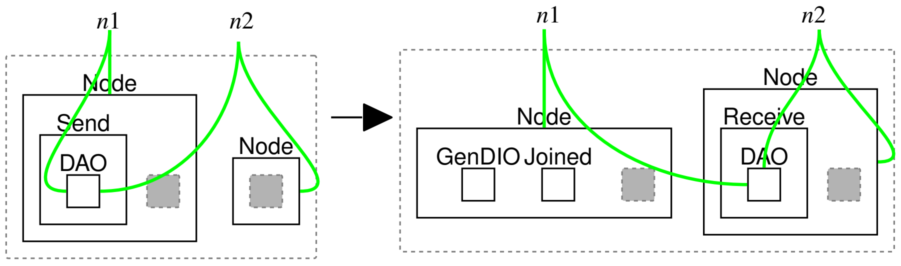
Step 4: Sending the DAO (Figure 5.7). Nodes register themselves by sending a DAO to the root. This is achieved by first applying the rule `sendDAO` (Figure 5.7a). If the receiving node is not the root (*i.e.*, has a PrefPrnt), the rule `sendDAOUpParent` is then applied to pass the DAO further up towards the root through the node's preferred parents set (Figure 5.7b). `clearDAO` (Figure 5.7c) 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).

In practice, this DIO would only be generated once a DAO-ACK is received from the root node. We assume that DAO-ACKs will always be successful, so we skip the steps that send them for now.

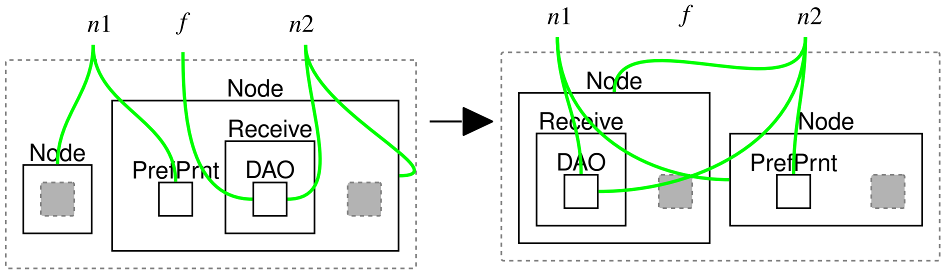
Once all the nodes have joined and fixed their ranks, no new DIO messages will be generated. At this stage, the DAG construction is complete and all the nodes will have joined at the minimum rank possible.

5.2 RPL Model Extension

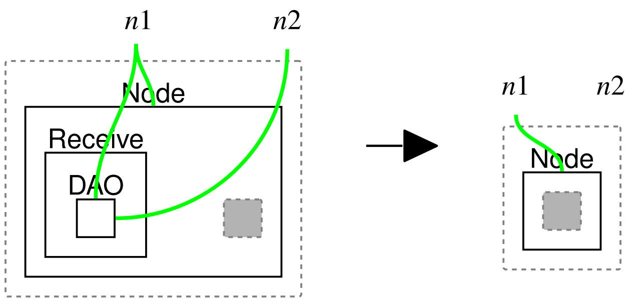
The bigraph model is flexible and open to extension, without requiring full implementation (*e.g.*, RPL implementations in simulators), by adding additional reaction rules as required. The



(a) `sendDAO`



(b) `sendDAOUpParent`



(c) `clearDAO`

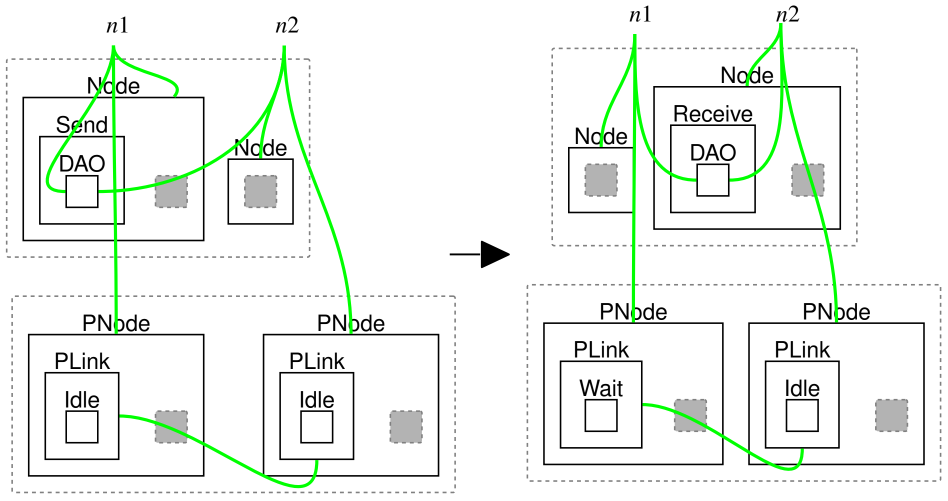
Figure 5.7: Reaction rules for sending the DAO.

provided RPL model above can be seen as a basis for a 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 [148, 67].

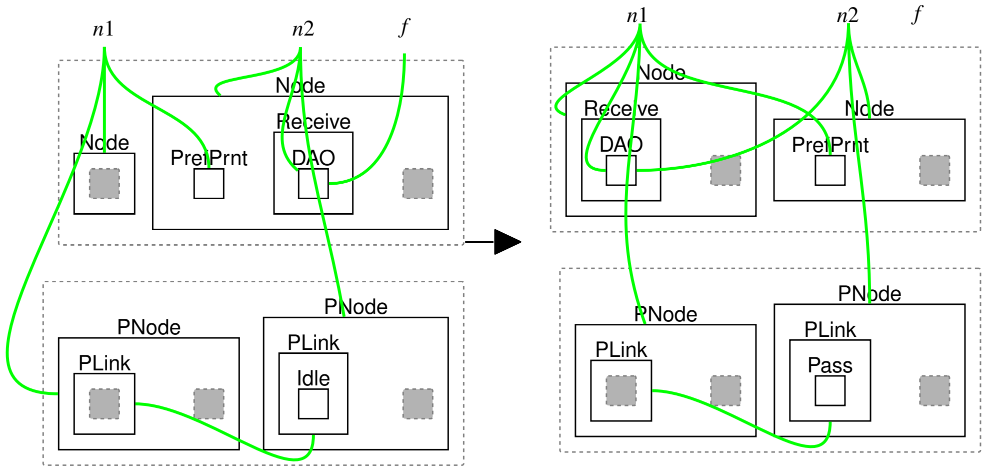
In the following sections, we show in detail how the initial model can be extended to include a DAO-ACK message, some of the security attacks and probabilistic extensions that show physical links and message failures.

5.2.1 DAO-ACk Modelling

The RPL initial model assumed the successful delivery of the DAO-ACK control message; therefore, it was not included in the model. In that model, the node is considered a fully RPL DAG node and, hence, starts to multicast DIOs after it replies with a DAO to the sender of the DIO, which it also sets as a preferred parent.



(a) `sendDAO`

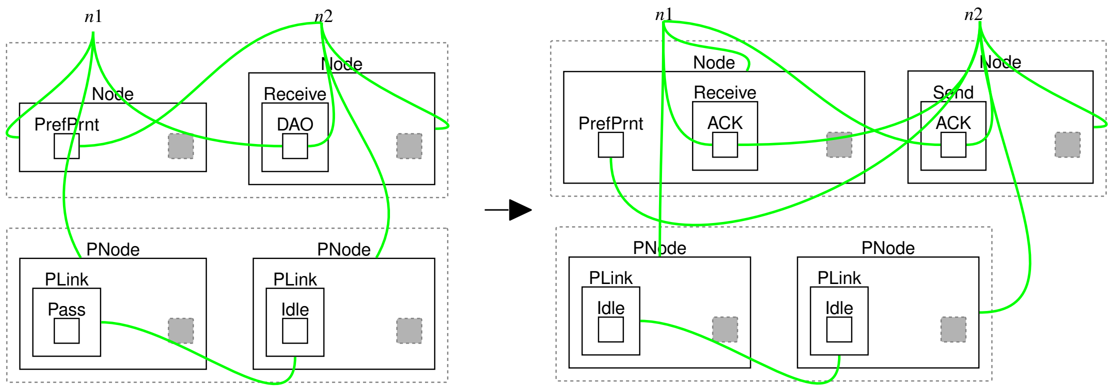


(b) `sendDAOUpParent`

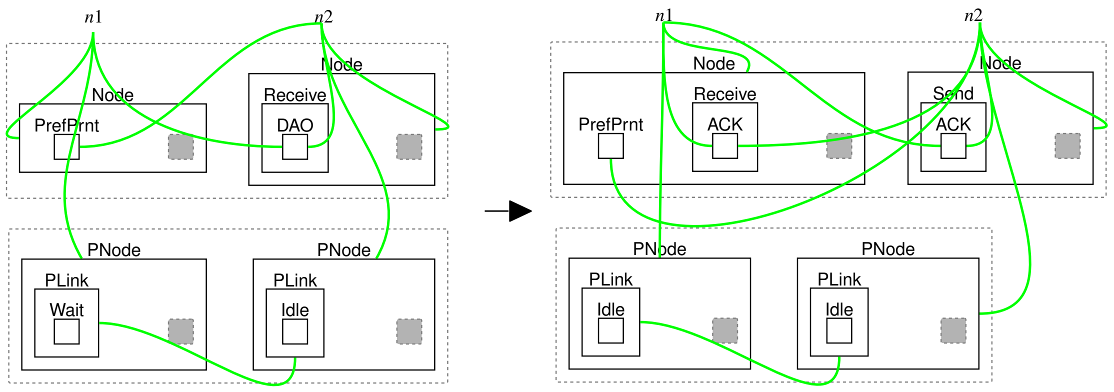
Figure 5.8: Updated reaction rules for sending the DAO.

However, to model the DAO-ACK message, the assumption of successful delivery is removed and the following scenario is modelled. A node that receives the DIO message replies with a DAO and waits for the acknowledgement to become fully an RPL node. To achieve this result, we conduct the following:

1. We introduce new statuses for the links; namely, Wait, for a node sending a DAO, and Pass for a parent that receives the DAO and passes it further up the network. As a result, the `sendDAO` rule is updated (Figure 5.8a) so that the node that replies with DAO waits for the acknowledgement by tagging its link with a Wait control rather than assuming successful delivery. Accordingly, it sets the sender as a preferred parent and fully joins the DAG – hence, it multicasts later upon receiving the DAO-ACK only.



(a) `sendACKRootParent`



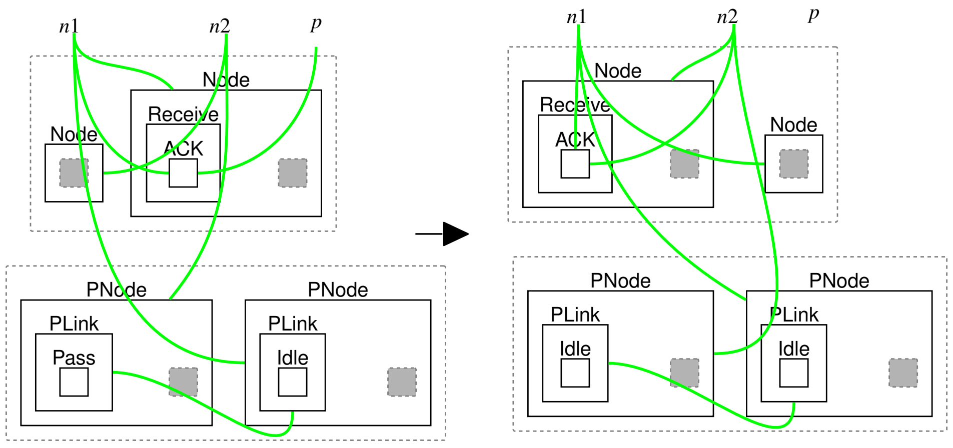
(b) `sendACKRootNode`

Figure 5.9: Reaction rules for replaying with the ACK.

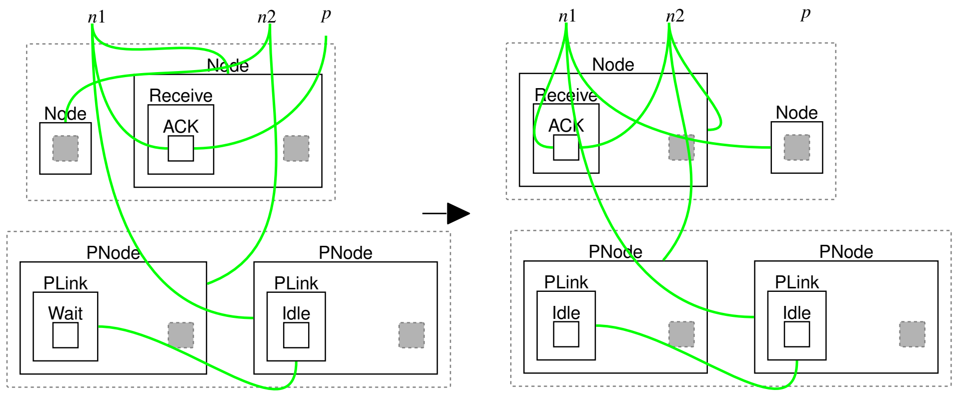
2. Similarly, the `sendDAOUpParen` rule is updated (Figure 5.8b), so that the set of preferred parent links are tagged with a Pass control for each node that receives the DAO and passes it up further to the root. Note that these controls are used to facilitate the tracking of the path of a DAO message that is sent from the prospective joining node to the root via the set of preferred parents. A node's link that is tagged with Pass, later reverts back to the Idle status when it passes the DAO-ACK messages downwards, as described below

3. Depending on the node that the root receives the DAO from, the root replies by one of the following rules:

   - `sendACKRootParent` rule (Figure 5.9a): The root receives a DAO from the last parent in the set of preferred parents of the prospective joining node. The root sends a DAO-ACK message to the sender of the DAO, which passes it down.
   - `ACK_root_node` rule (Figure 5.9b): The root receives a DAO directly from the prospective joining node. That is, the prospective joining node is a direct child of the



(a) `sendACKParentParent`



(b) `sendACKParentNode`

Figure 5.10: Reaction rules for passing the ACK.

root, hence the root replies with a DAO-ACK to it directly and, hence, the node joins.

4. Depending on the parent, in a preferred parent list, that receives the DAO-ACK in the first rule of step 3 above, the following rules apply:

   - `ACK_parent_parent` rule (Figure 5.10a): The node is a parent in the preferred parent list; hence, it sends the DAO-ACK downwards to the next parent.
   - `ACK_parent_node` rule (Figure 5.10b): The parent is the direct parent of the prospective joining node; hence, it sends the DAO-ACK to the node that is waiting for it directly.

5. One reaction rule, `toJoin`, is encoded to enable a node to join the existing DAG once it receives the DAO-ACK. GenDIO is introduced here so that the node multicasts DIOs at this point to allow more nodes to join, as described in Section 5.1.3.

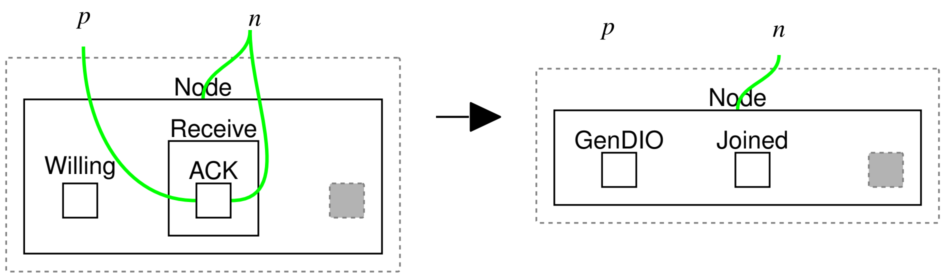
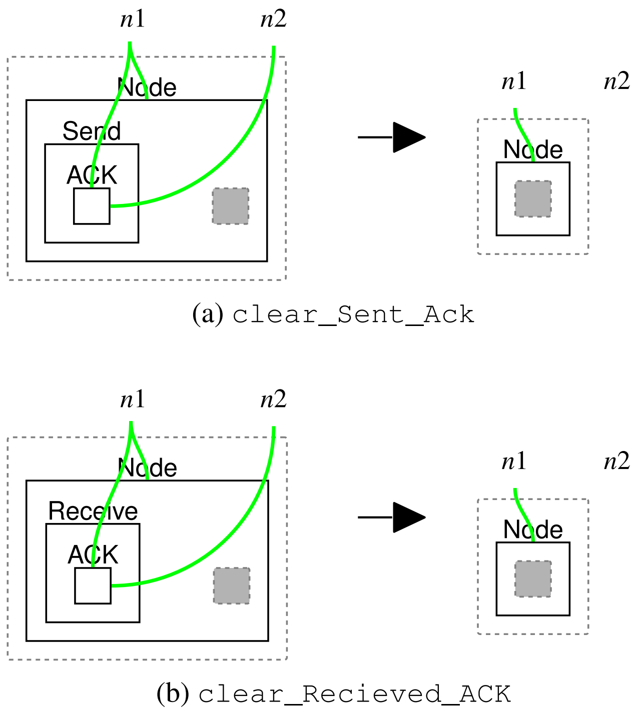


Figure 5.11: Reaction rule `toJoin`



(a) `clear_Sent_Ack`

(b) `clear_Recieved_ACK`

Figure 5.12: Reaction rules for clearing the DAO-ACK.

6. `clear_Recieved_ACK` and `clear_Sent_Ack` rules are added to clear the DAO-ACK message (Figure 5.12).

5.2.2 Security Extensions

As seen, the bigraphical RPL model is open to extension, which also facilitates the security verification process. The security extension is achieved by simply encoding the required rules that model an attack. We assume that the reaction rules for attacks have 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. We encode *a rule* with a randomly selected node, other than the root, to be a compromised node using a Python script, given in Section 5.3.1. This will then be used in the following attacks.

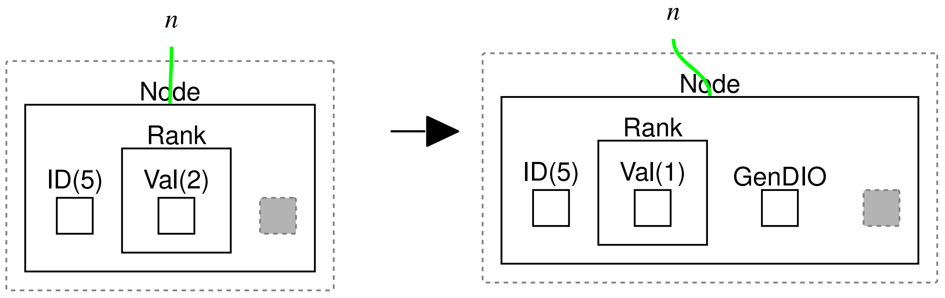


Figure 5.13: Reaction rule `sinkholeAttack`: node 5 advertises a fake rank.

1. Sinkhole Attack

In this scenario, a compromised node advertises a fake rank to get the neighbours to connect to it. We model this by using a reaction rule, as in Figure 5.13, that gives a (randomly selected) node (5 in the figure) rank 1 and requests for a DIO to be sent by generating a GenDIO on the right-hand side. When other nodes receive the DIO they will update their rank accordingly and they will assume that there is a better path to the root.

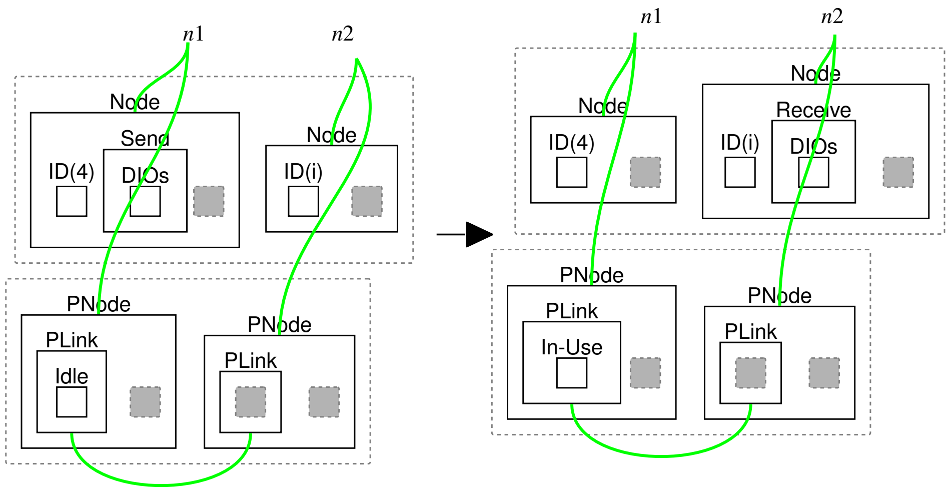


Figure 5.14: Reaction rule `helloFloodAttack`: node 4 continually multicasts DIOs

2. Hello Flood Attack

In a hello flood attack, a compromised node continually multicasts DIOs to its neighbours in an attempt to fill up Receive buffers and cause nodes to be so inundated with message processing that they cannot forward data to the root. We can model this type of attack with a rule, like in Figure 5.14. To explicitly model failures, we would also need a trivial reaction rule that marks a node as “Failed” when it contains a given number of DIOs.

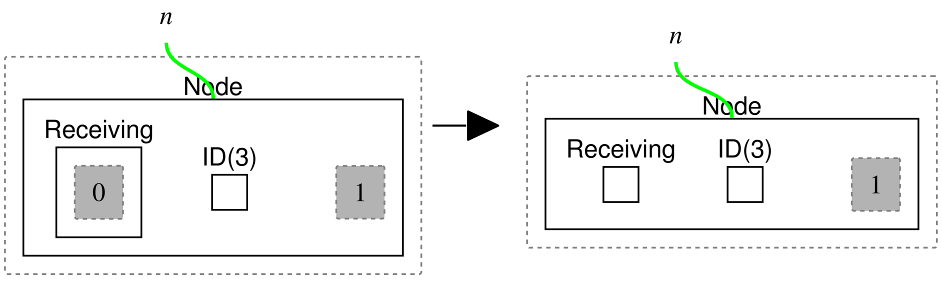


Figure 5.15: Reaction rule `blackholeAttack`: node 3 drops incoming messages

3. Blackhole Attack

In a blackhole attack, a compromised node simply drops all the packets it receives to cause network disruption. This can be modelled using a simple reaction rule, like in Figure 5.15, that removes any message from the Receive buffer of a compromised node (3 in the figure).

Importantly, we are not tied to a single attack and we could run all of these possibilities in parallel, *e.g.*, the protocol, or proposed security mitigations, may only be robust to one attack at a time.

5.2.3 Probabilistic Extension

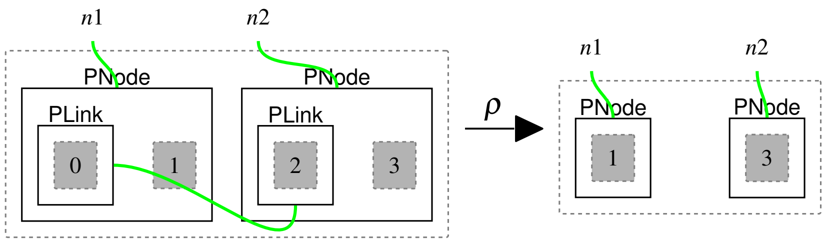


Figure 5.16: Stochastic reaction rule `dropLink` with rate ρ.

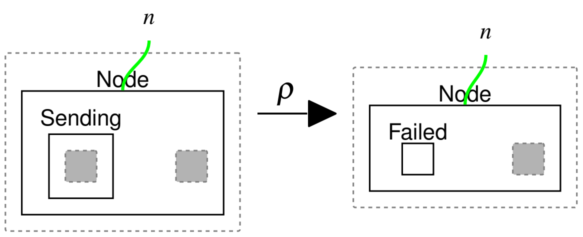


Figure 5.17: Stochastic reaction rule `msgFailure` with rate ρ.

We show how to extend the bigraph model to include message failure or variability in the radio signal due to interference or other environmental factors.

1. Links Drop

We add a *stochastic* reaction rule [75, 9], as in Figure 5.16, 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[4].

2. Message Failures

Another *stochastic* reaction rule is added (Figure 5.17) to show message failures with rate $\rho \in \mathbb{R}_{>0}$. We can also specify a reaction rule for each message, *e.g.*, DIO, to show a particular message failure.

5.3 Discussion

This chapter shows how Milner's bigraphs provide a formal model for RPL and the belief is that the same approach is applicable to other similar routing protocols. Considering the characteristics of WSNs, *e.g.*, radio signal variability and failures, and the features of bigraphs formalism, we note our observations on the utilisation of bigraphs for modelling and analysing routing protocols in the subsequent sections.

5.3.1 Model Generalisation

During the modelling process, we aim for our model to be applicable to any topology with n connected nodes. To ensure the achievement of this goal, we develop a Python script that generates a random valid topology in the BigraphER syntax based on an input number of nodes entered by the user. There are no other requirements for the generated topology to be valid against our RPL model. The demonstrated RPL model (without a predefined initial state) is used as an input in the RPL model template. The script then updates this input file with the generated topology placed as an initial state in the model each time it is run. In addition, the template can include the extensions (Section 5.2) by adding the required rules, and it can also produce more features, *e.g.*, selecting a random node to be the compromised one for the security rules, with each generated topology. More details relating to the Python script are provided in the next chapter. This ability of bigraph models to be easily updated with different topologies allows for a quicker experiment for several scenarios, which leads to more robust validation results.

5.3.2 Bigraphs Usability

The usability of formal methods is a pressing question in protocol design, with the Internet Engineering Task Force (IETF) recently proposing their *Usable Formal Methods Research*

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

Group[5]. We believe that bigraphs, in particular, offer some features related to the usability as presented in the following.

RPL Standard

Bigraphs as a graphical formal modelling formalism provides an unambiguous and precise description of protocols. By reviewing the RFC6550 standard [5], focusing on the core function of the RPL, *i.e.*, constructing valid DAGs, we found that modelling a protocol from its specification is a challenging process. RFC6550, in particular, is a long and complicated document that consists of several issues. The following are some of these issues that are observed alongside others [70]

- The RPL standard comes with several complementary documents, *e.g.*, RFC 6552, RFC 6719 and RFC 9035. Although these numerous documents aid in the understanding of the standard specification document, they interrupt the flow and impede readability.
- Many features are left optional, leading to various interpretations and, hence, implementations.
- Even though there is a separate document that explains the words that are used to indicate the necessity level of a feature (*e.g.*, must, should, may, required, recommend and shall), the variety of these terminologies affects the readability of the document and increases the standard complexity.
- Despite the importance of building a valid route mechanism in RPL, the definition of control messages and the description of the DAG construction are spread throughout the document. The standard provides the static formats of the messages with no obvious description of their dynamic flow.

User-Defined Rules

The expressiveness of bigraphs, combined with user-defined rules, allow for customisation of the modelling process to model complex systems. That is, modellers can clearly express their models, which leads to an enhanced expressiveness and flexibility of the model. Furthermore, by customising the rules, the modeller can use an expression that is easier to reason about, *i.e.*, bigraph patterns (Section 4.4.5). In addition, user-defined rules facilitate the model reusability when appropriate.

Model Extensions

A bigraphical RPL model is open to expansion by adding essential controls, encoding required functions as rewrite rules and updating the existing rules when necessary, as shown in Section 5.2.

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

Table 5.2: RPL model extensions.

Extension	Controls	Rules	Updated rules
RPL main model			
Initial models	24	16	–
DAO-ACK model	+3	+7	2
Attacks			
Sinkhole Attacks	0	+1	–
Helloflood Attack	+1	+1	–
Blackhole Attack	0	+1	–
Probablistic			
Link failure	0	+1	–
Message receive failure	+1	+1	–

We summarise the changes made to different extensions in Table 5.2. As illustrated, the majority of extensions require adding a single rule with no extra controls or updating any other existing rules.

Consequently, we strongly believe that involving formal modelling is essential throughout the entire process of designing a protocol. As a result, the protocol can be progressively developed, evaluated, refined and documented in a more cost and time efficient manner.

5.3.3 Model Characteristics

The presented RPL model successfully covers the main attributes that a verification technique should consider for routing protocols. As an illustration, the features of the RPL bigraph model align with the characteristics described in Section 3.6 as follows:

- Using perspectives (Section 4.4.4), the RPL model relates nodes both spatially (Physical perspective) and with non-local connectivity (Routing perspective), as Figure 5.1b presents.
- It shows complex user-defined rules. This is to enable modelling of both locality and connectivity, *e.g.*, Figures 5.4a and 5.8a.
- By using probabilistic bigraphs, the model shows the ability of modelling the dynamic changes of the RPL *e.g.*, loss of packets (Section 5.2.3).
- As shown in Section 5.2, the model is flexible and extendable without complex re-implementation. This supports multiple attempts to discover the effects of protocol changes.
- The RPL bigraphical model can be applied to any connected topology, irrespective of the number of nodes, as illustrated in Section 5.3.1. This enables a generalisation of the verification results.

- As shown in the reaction rules above, which are represented diagrammatically, the model can be easier to understand compared with other mathematical-based formalisms for a non-specialist audience. This includes protocol designers who may be unfamiliar with the complex notation that is often found in formal modelling tools.
- Bigraphs do not have a clock notion, so the model does not consider the RPL timing aspects. However, since bigraphs are an extensible formalism, we investigate bigraph-modelled real-time systems (Chapter 7).

5.3.4 RPL Time Aspects

As mentioned above, bigraphs are ineffective when modelling real-time systems. Since bigraphs do not directly support timers at present, we do not consider RPL timing features here. Although this does not impact the modelling of valid RPL DAGs, we choose to enhance bigraphs by incorporating clocks, which will result in more realistic models. Therefore, we propose a modelling technique that enables bigraphs to model and verify real-time features that routing protocols generally contain. We then apply this technique to the RPL initial model by extending it, as illustrated later in Chapter 7.

5.3.5 State Space Explosion

The characteristics of WSNs, such as a vast number of nodes and high network density, indicate the likelihood of encountering the state-space explosion problem. In our model, we implement the following techniques in order to minimise the state space as much as possible:

- Abstraction techniques. We simplify the model by considering some assumptions, such as a node receives one DIO message at a time.
- Reduction techniques. We focus on eliminating unnecessary states and paths by using instantaneous rules for some functions, such as rank calculations.
- Cumulative modelling. We build the essential model as a first step and then expand it by adding more functions, as required by specified properties.

5.4 Summary

This chapter demonstrated how bigraphs could be leveraged in routing protocol formal modelling. Using only 24 entities and 16 reaction rules, bigraphs model the initial routing DAG construction steps of RPL – a popular protocol for wireless sensor networks, as illustrated in Section 5.1. Bigraphs are open to extension by simply adding a few additional reaction rules, rather than requiring a full implementation effort (Section 5.2). The chapter denotes, in Section 5.3, that using bigraphs to model WSN routing protocols provides a model that is compatible with the required features in the proposed modelling formalism for WSN protocol modelling. It also shows that using bigraphs improves the usability of protocol standards as an extendable diagrammatic notion. Bigraphs permit the application of some of the state space mitigation techniques while modelling the protocols, leads to a minimisation of the state space for future model verification.

We believe that other routing protocols, *e.g.*, RIP [121] or LEACH [52], could be modelled using a similar approach by utilising common features between protocols. We also believe that modelling multiple protocols could beneficially determine a set of common approaches for modelling protocols in general.

While this chapter presents the formal model of the RPL using bigraphs, the next chapter formally verifies the RPL model using model checking. It also compares the bigraphical RPL model against RPL-Lite, the default Contiki-NGs RPL implementation which is the most notable RPL one.

Chapter 6

Routing Protocols Analysis

Reviewing the state-of-the-art in RPL validation and verification reveals that the most widely used approach is simulation. A formal modelling approach is leveraged in a very limited number of studies, *e.g.*, [110, 2]. Therefore, the main motivation behind this chapter is to offer protocol designers a broader view of alternative protocol verification approaches. To show this, a comparison between simulation (the most popular approach) and formal method (a promising technique) is performed. The comparison focuses on the validity of the constructed DAGs and the time required by the two approaches to find them. Due to the popularity of the Cooja simulator in RPL-related research, this thesis makes use of it in the comparison experiments.

After providing a brief review of the RPL analysis techniques in Section 6.1, Section 6.2 presents the formal approaches used for verifying RPL bigraphical models. Next, a detailed description of the experiments and the results obtained is illustrated in Section 6.3. Section 6.4 demonstrates a comparison between RPL presentations in both bigraphs and RPL-Lite documents. The chapter discusses the findings in Section 6.5 and concludes in Section 6.6.

6.1 RPL Analysis Approaches

Routing protocol for low-power and lossy networks (RPL) has gained significant interest. Since it is standardised, it has been cited 3551 times[1]. Numerous articles assess it, evaluate its performance and proposed solutions for the existing challenges and limitations, *e.g.*, [99, 4, 79, 44]. Simulation is the most popular approach used to analyse RPL and evaluate RPL enhancement proposals. RPL has been implemented in various simulation environments, such as Ns-3 and Cooja. The latter is the most applicable simulator for RPL-related studies [35].

Cooja is a popular emulator for IEEE 802.15.4 networks with devices running Contiki-based firmware to simulate the radio environment. Contiki-NG [107] is an operating system specifically designed for devices with limited resources in the Internet of things environment. ContikiRPL is a reliable real-time implementation of RPL, in which RPL is fully supported. Cooja offers

[1]as 09-12-2024

multiple features, such as computing the energy consumption and offering a TimeLine module that visualises the energy and the network traffic [79].

Despite the popularity of the RPL protocol, there is a very limited number of works that formally model it. A recent paper [2] employs coloured Petri nets (CPNs) to model the security schemes of a subset of RPL that covers a single instance to validate and verify both secure and non-secure modes of RPL protocol. The authors model RPL control messages – focusing on the DIO and DAO messages since they enable interaction with the other nodes – and evaluate the existing security standards for RPL through a state-space analysis using a CPN-based simulator. The article highlights that using formal techniques to model large-scale complex systems could reduce its clarity and comprehensibility, where it uses CPNs to alleviate this issue by offering a compact representation of the system model. This thesis argues that using bigraphs reduces this limitation of formal approaches by exploiting bigraphs characteristics *e.g.*, diagrammatic notion, open to extension, etc. On the other hand, this thesis shows agreement that using formal approaches leads to the well-known state space explosion which they alleviated by using hierarchical coloured Petri nets since CPNs assists in constructing modules of large-scale systems that offer a compact representation of the system model. The next section presents the alternative techniques, static analysis, used in this thesis to mitigate this issue. Unlike our approach (Section 6.2.1), theoretical analysis is utilised to verify the correctness of the DAG construction and maintenance in a manner that does not consider specific objective functions and routing metrics using LTL to specify the properties [110]. The procedure of the latter work provides additional formal conditions, in addition to what exists in the RPL specification, to offer an important understanding of the requirements of the objective functions and routing metrics. It then converts theoretical findings into practical guidelines for RPL adopters to improve the reliability of RPL deployments. In the next section, we show our approaches for verifying RPL bigraphs model.

6.2 RPL Model Verification

Due to our observations that reveal a positive correlation between the well-known state explosion issue and the number of nodes, where an the increase in number of nodes corresponds with an increase in the state space, we use the automatic technique, *i.e.*, PRISM to verify our model on small topologies. However, we employ a static analysis approach to prove that our model verifies the RPL protocol regardless of the number of nodes and topology. That is, we consider two approaches for verification: model checking (over a given topology) and static analysis of the reaction rules to prove that the properties hold regardless of the specific topology.

6.2.1 Model Checking

Model checking is useful here for small topologies due to the state-space explosion issue, which is common in formal verification, and benefits from automation. Furthermore, the support for probabilistic reasoning can also be utilised, *e.g.*, to model real link failure rates.

To formally check the correctness of DAGs, we specify a set of properties to verify our RPL model through PRISM. The latter is an open-source automated verification tool that provides model checking for different types of probabilistic models, such as discrete time Markov chains (DTMCs). Although our initial 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) [30]. The choice of PRISM is also motivated by technical reasons, *i.e.*, BigraphER currently only supports PRISM output format. However, this could also be a benefit since it allows probabilistic modelling within RPL and other protocols, *e.g.*, assigning probabilities/rates that some links fail (as shown in Section 5.2).

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

We use NetworkX [2], a Python package for the creation, manipulation and analysis of graphs, to assist in generating a set of properties to validate the model. This is because the current definition of bigraph patterns does not include spatial modalities such as, for instance, $\mathscr{R}$ (*reachable from*) in SLCS [29, 87] that would allow reasoning on the path properties of 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 in a set of CTL formulae in PRISM format. In essence, this allows us to check that our local protocol creates the correct global DAGs.

Properties for Verification

The following temporal properties check that the properties hold for all possible paths in the transition system. We use n to indicate the number of nodes in a topology.

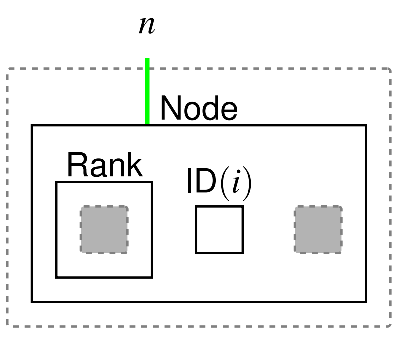
Property 1: All nodes eventually join the RPL DAG. Since we assume fully connected physical topologies, RPL should eventually find a path between the root and all the nodes. We check this by assigning a label $\texttt{joined}_i$ whenever a node i is assigned a rank and so becomes part of the DAG (Figure 6.1a).

To check that all the nodes eventually join, we use the following *family* (*i.e.*, one property per i[3]) of properties:

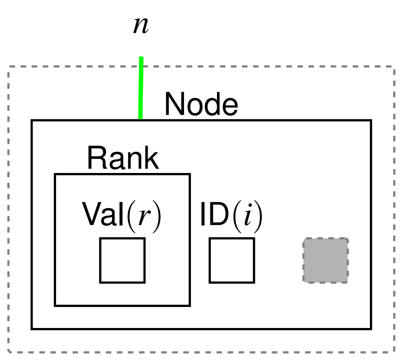
$$\forall i.\, \mathbf{A}[\mathbf{F}\,\texttt{joined}_i], \quad \text{where } 1 \le i \le n. \tag{6.1}$$

[2]`https://networkx.org/`

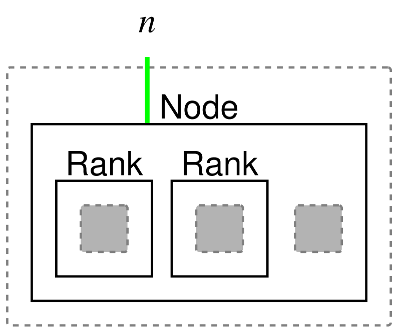
[3]To verify that we can enumerate these properties, we ensure i etc. is drawn from a finite set of identifiers.



(a) `joined`$_i$: node i joined the RPL DAG.



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



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

Figure 6.1: Bigraph predicates to check RPL properties.

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 the 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 Figure 6.3b and Figure 6.3c, node 4 will always eventually have rank 2 even though there are (at least) two different valid routes to it. Here, we check that the nodes eventually join their optimal rank.

To write this property, we obtain 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 (Figure 6.1b). To check that each node joins with optimal rank, we then use the (family of) properties:

$$\forall i.\mathbf{A}\left[\mathbf{F}\,\texttt{rank}_{i,r}\right], \quad \text{where } 1 \leq i \leq n,\ r = sp(i). \tag{6.2}$$

Since this checks all the 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 need to check that the DAG created by RPL is always actually acyclic. The mechanism RPL uses to achieve this is via the ranks. We can guarantee cycle freedom by ensuring that the nodes only ever join on a single rank (otherwise a node could be a descendent of itself).

We undertake this task by using the labelling `multijoin` that matches using the bigraph shown in Figure 6.1c, *i.e.*, two ranks and anything else. We can then check that this *never* occurs, *i.e.*:

$$\mathbf{A}\left[\mathbf{G}\,\neg\texttt{multijoin}\right]. \tag{6.3}$$

A quantitative analysis on the extended model can be carried out with PRISM by expressing the properties using continuous stochastic logic (CSL) [12] instead of CTL. For example, the property $\mathscr{P}_{\geq p}\left[\mathbf{F}\,\texttt{joined}_i\right]$ holds if node i (with $1 \leq i \leq n$) eventually joins the RPL DAG with probability greater or equal than p.
Additionally, a security analysis can be performed on the extended model which includes attacks. For example, a reaction rule is encoded to detect multiple receptions of the DIO message. Subsequently, another rule is applied to block the node from receiving further messages. This can then be verified by specifying the following property:

$$\mathbf{E}\left[\mathbf{F}\left(\texttt{attackDetected}\right) \wedge \left(\mathbf{X}\,\texttt{attackBlocked}\right)\right]. \tag{6.4}$$

6.2.2 Static Analysis

While the model checking approach is useful because 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 and the number of nodes, the following properties must hold for the application of all the possible (and applicable) reaction rules. Proper execution of the rules is guaranteed by using priority classes that provide an ordering on the rules.

Properties for Verification

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 that this was proven for a given topology with the model checking in Property 1.

For the base case, we show that any child of the root that is not already in the DAG will eventually join it, *i.e.*, the child node obtains 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 described in Figure 5.4 and terminating with an application of the rule `clearMulticastEnd`. Although different interleavings are possible since children can be processed in different order, this step of rewriting is confluent. Since we assume that 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. This 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, will eventually join 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 Figure 5.5 (DIO handling) and Figure 5.6 (rank computations) can be interleaved as the parent node can also transmit to connected nodes already in the DAG, where we ensure nodes selecting the possible minimum ranks (Property 2 above).
- rule `incRank`(n) with $n > 0$ 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 the pattern `multijoin` in Figure 6.1c never occurs on the right-hand side of any reaction rule in our model, we can never reach a state in which two Rank entities are within a Node at the same level of nesting.

6.3 RPL Formal Verification and Comparison to Simulation Approaches

This section presents a comparison between the initial RPL model and RPL-Lite, the default Contiki-NG's RPL implementation[4]. First, the modelling assumption listed in the previous chapter for the RPL bigraph model is recalled as follows:

1. We only model downwards 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 sufficiently short relative to the message transmission time, and there are no packet drops in the network. Note that, because we undertake full model checking, all the 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 the DAG construction.

Note that these assumptions are appropriate when comparing against RPL-Lite since:

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

1. The timing aspects (*i.e.*, clocks) of RPL-Lite are not considered because 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 compared with 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 the orderings, we will obtain all the possible parents regardless of the link quality or the metric employed. One way to view this is that all possible link quality alternatives are considered.

6.3.1 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.

1. Automatic Model Generation

A Python script is developed to generate fully connected topologies, *i.e.*, there will always be a path to all the nodes without self-loops. The code accepts the required number of nodes as an input and produces topologies in *both* formats: Cooja (as a `.csc` file) and BigraphER (as `.big` file). The script may also provide other modelling features, *e.g.*, randomly select a node, other than the root, to become a compromised node that is then inserted into an attack rule. Code that generates the topologies and reproduces the experiments is available online [126].

Figure 6.2 shows the experiment flow chart for *each* topology, while an example of a five-node topology is shown in Figure 6.3a.

For each topology, the following are performed:

1. Encode the generated topology for both Cooja (`.csc` file) and BigraphER (`.big` file).
2. The generated topology is inserted into given templates: `.csc` file as it is provided by Cooja and `.big` as it is described in Chapter 5.
3. Run the model and simulation and extract the DAGs constructed by both BigraphER and Cooja. The DAGs generated by BigraphER are obtained by parsing all the deadlock states

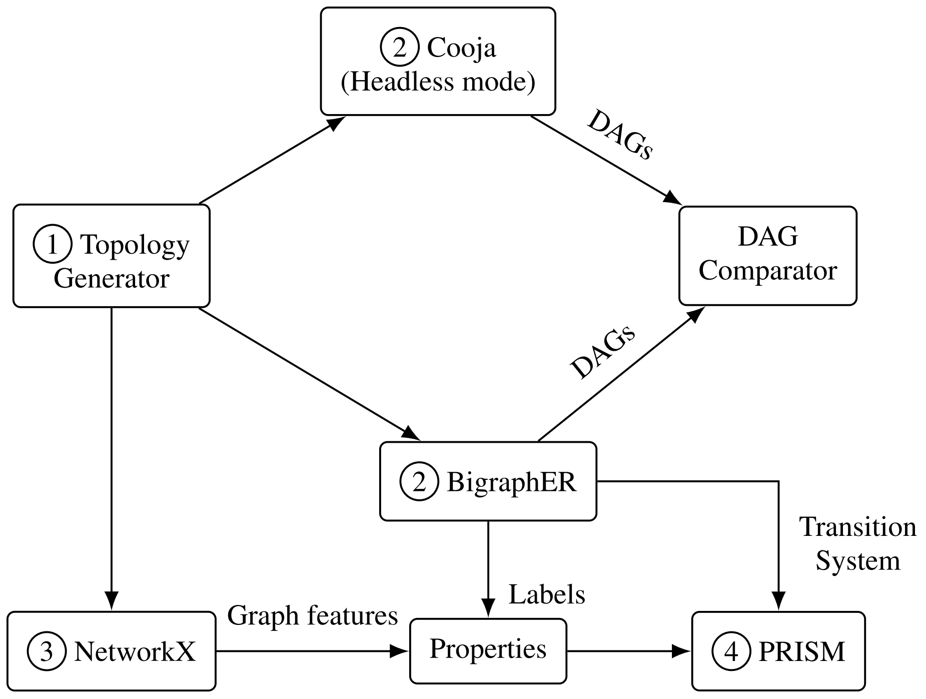


Figure 6.2: Experiment approach.

in the transition system, while those found by Cooja are constructed by parsing each simulation log file.

4. We check if the two DAG sets are the same (often Cooja misses possible DAGs, which BigraphER finds).

5. We then use NetworkX to generate a list of the predefined properties. NetworkX extracts graph features, *e.g.*, the given graph (DAG) has no cycle, which we use to validate the generated DAGs of the bigraph model against the properties list shown above.

6. We verify, with PRISM, each property (given above) against each DAG constructed by our model. The model-checking results are saved in a text file. We assume that the DAGs found by Cooja are correct; therefore, we do not validate them.

2. Experiments Run

We performed our experiments using GNU Parallel [138] on machines with dual 8-core Intel Xeon E5-2640v2 CPUs (2 GHz; no hyper-threading) and 64 GB 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 a 1000x simulation speed, and terminates when the simulator constructs a DAG comprising all n nodes.

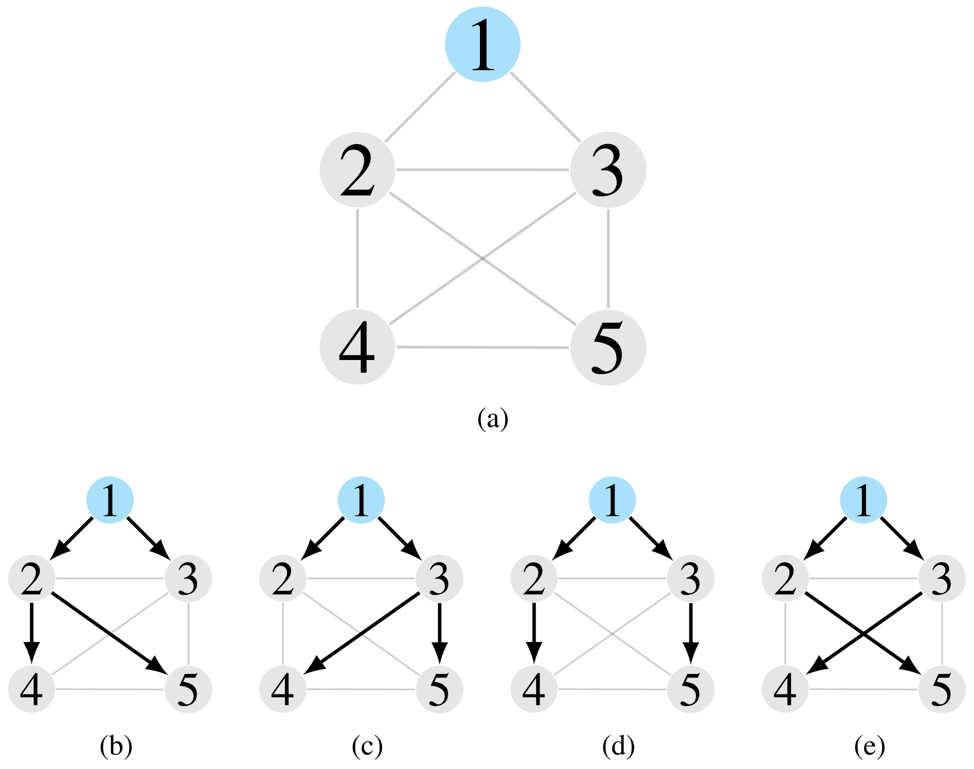


Figure 6.3: (a) Example physical topology and (b)-(e) all the valid RPL DAGs .

6.3.2 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 sufficient for discovering all the possible DAGs) and 500 runs of Cooja for each random topology.

1. Comparing DAGs Found

A key benefit of the model checking approach of bigraphs is that a single run finds all the valid RPL DAGs, while a single run of Cooja finds 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 [126].

Number of found DAGs

Figure 6.4 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 the cases (*i.e.*, 59/100) and, in some instances, these differences can be quite startling: in one instance, BigraphER detected 14 more valid DAGs compared with Cooja. Even after increasing to 1000 runs, no additional DAGs were found by Cooja.

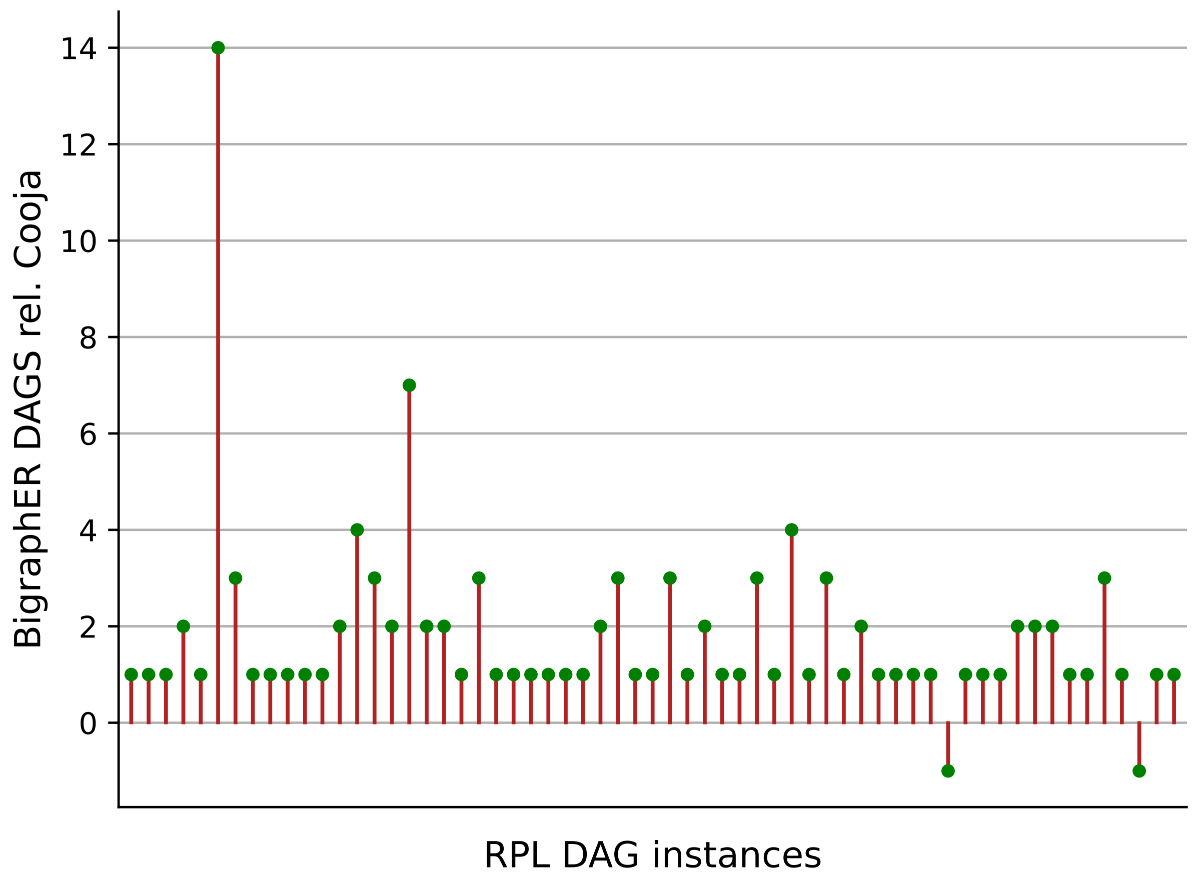


Figure 6.4: Number of RPL DAGs found with BigraphER and Cooja: above 0 implies BigraphER found *n more* DAGs than Cooja. Topologies where BigraphER and Cooja agree are not shown (39% of cases).

There were two cases where BigraphER times out (*i.e.*, $>$ 24 hours runtime). Since we do not currently support partial output that shows us how many DAGs are discovered before this occurs, we utilise BigraphER simulation to support our analysis. In one case, BigraphER finds all the 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 that Cooja had enough time to find interesting RPL DAGs.

Liklihood of Found DAGs by Simulation

In Figure 6.5 we show, for all the 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 the runs; often in 80% of the runs. Finding a third or fourth DAG occurs 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 resources if we want to be confident of finding all the cases. Bigraphs do not have this limitation.

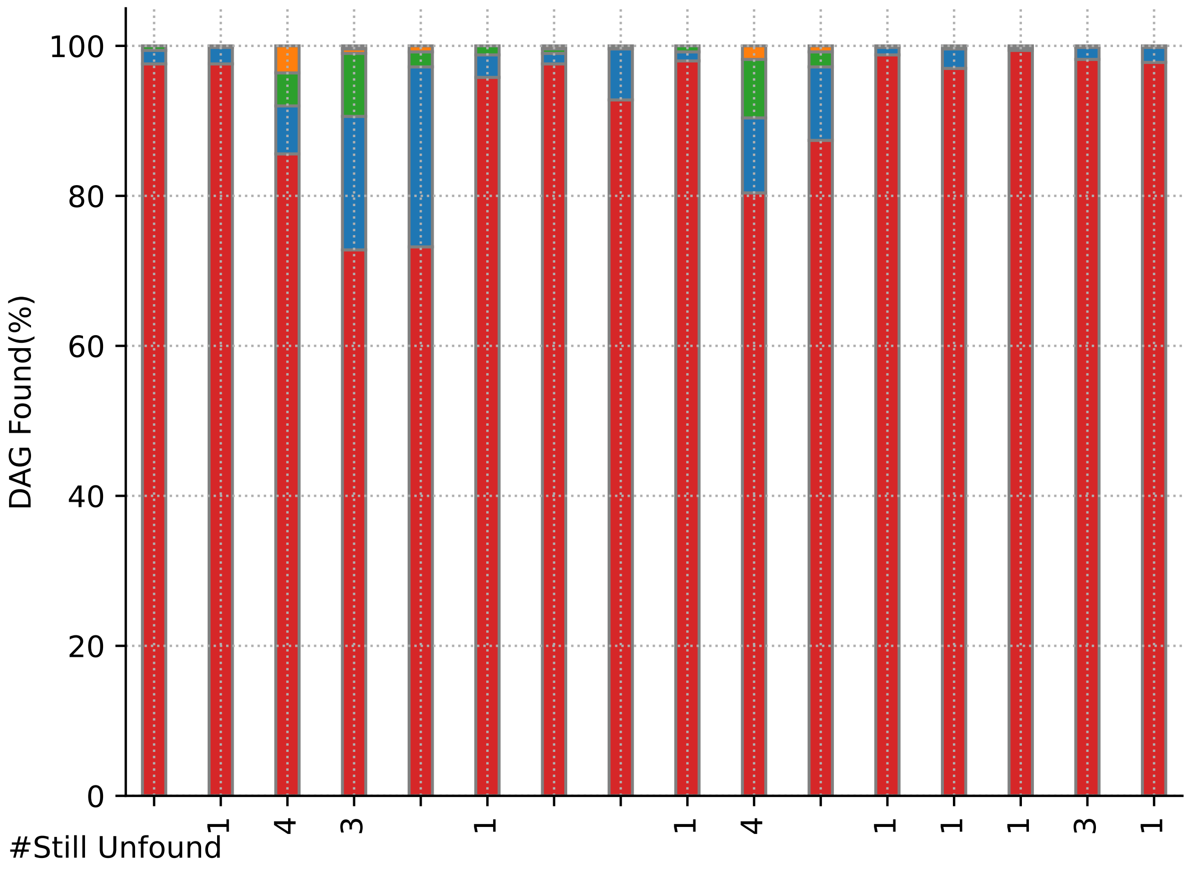


Figure 6.5: Distribution of RPL DAGs within Cooja simulations (for the 16 instances that find 3 or 4 DAGs). Colours indicate the different DAGs found; *i.e.*, the red colour indicates the most DAG repeatedly found in 500 simulations. The *x*-axis labels show instances where more DAGs were possible but were never discovered even after 500 simulations.

2. Running Times

We have shown that BigraphER finds more valid RPL DAGs than Cooja in most cases. As Figure 6.4 indicates, Cooja finds the same number of valid RPL DAGs as BigraphER in only 39% of cases. This highlights the issues of simulation missing outputs when analysing protocols.

There are trade-offs in the time required to discover all the DAGs (bigraphs) instead of a simulated subset of the DAGs (Cooja). Figure 6.6 shows the distribution of BigraphER and Cooja runtimes. Here, bigraphER runtimes are clustered towards the left-hand side showing that, in most cases (*i.e.*, 83/100), we can find all the RPL DAGs in less than 60 minutes. Of the remaining instances, most complete within 180 minutes (3 hours) but 2 of them take longer than 1 day to compute (so timed 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 have much less of a spread and a mean (for 500 simulations) of 154 minutes (2.5 hours).

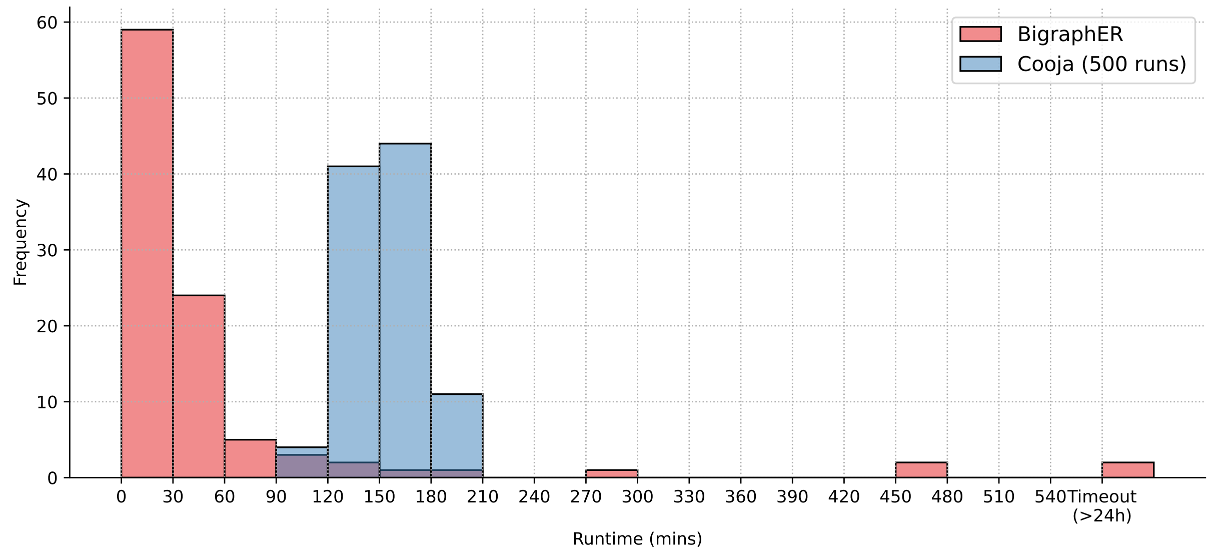


Figure 6.6: Distribution of BigraphER and Cooja runtimes. Timeout is 24 hours.

6.4 RPL Presentation in Simulation and Bigraphs

Cooja uses Java classes separated into multiple files; hence, tracking and debugging the code is difficult. Furthermore, since the documentation of the implementation is written in English, it leads to ambiguity and misinterpreting. On the other hand, bigraphical models are graphical descriptions, which makes them easier to read and track even for non-expert users. In the following, we compare the two approaches by considering the DAGs construction process. Note that we consider the implementation document rather than the implementation itself, as it is a challenging task due to its spread and length. We provide a description here for the parts that we model from the RPL-Lite documentation, followed by the corresponding bigraph model from the previous chapter (Chapter 5.

Send DIO

When a node starts running as the DAG root (whether it is border router or not), it will advertise the DAG with DIOs (DODAG information objects).

A description of the corresponding bigraphs is given in Figures 5.3 and 5.4

Handle DIO

Whenever hearing a DIO, the node might choose to join the DAG. If it joins, its first task is to select an RPL-preferred parent.

A description of the corresponding bigraphs is given in Figure 5.5a

---

Send DAO

Once a preferred parent is chosen, a node then registers itself through a DAO (destination advertisement object).

The corresponding bigraphs are given in Figure 5.5a

Send DAO-ACK

Upon receiving the DAO, the root will add the node to its routing state: it will store the child-parent relationship, which is used later for source routing. In RPL Lite, by default, the DAO messages have the $'k'$ bit set, which means that they must be acknowledged by the root.

The corresponding bigraphs are given in Figures 5.8, 5.9 and 5.10

Fully Joined

After receiving a DAO-ACK, the nodes know they are fully part of the network and are reachable. Only then will they start advertising DIOs in turn and let more nodes join, forming a multi-hop mesh network.

The corresponding bigraphs are given in Figure 5.11

6.5 Discussion

As seen above, simulation does not find all the possible valid DAGs; hence, it does not provide full coverage. This has a negative impact when it is required to ensure the validity of every single path, *i.e.*, guaranteeing the absence of faults or vulnerabilities. On the other hand, bigraphs find all the possible valid routes, which guarantees the correctness of the protocol. However, due to an explosion of the state space, bigraphs need a considerable amount of resources, *e.g.*, time.

In short, the differences between Cooja and BigraphER DAGs have implications for future protocol analysis: it is clear neither is sufficient on its own. Full model checking can show many more cases, which increases confidence that the protocols work in a much wider range of environments. However, the presence of state-space explosion issues may make bigraphs impractical to use, meaning that simulation can provide additional insights in such cases. There could be a point where the emulation capabilities of Cooja and the modelling of bigraphs may unite to provide interesting insights, *e.g.*, first letting Cooja generate the most common topology and then exhaustively checking security via bigraphs.

It could be said that a key benefit of the formal approach is that we obtain all the possible DAG construction paths within a single run. Although this can take a significant amount of time, this allows us to check that the protocol works in a much larger set of situations. Given the increased number of DAGs found and the fact that BigraphER usually manages to find these in less than 1 hour, the modelling approach seems to be a valid alternative to Cooja. However, as mentioned, we believe that the best approach is for both techniques to be used in tandem.

6.6 Summary

This chapter presents a concise review of the two widely used approaches in verifying RPL, namely, simulation and formal methods (Section 6.1). Then, Section 6.2.1 outlined the interesting properties that are verified against the initial bigraphical RPL model using a PRISM model checker on predefined topologies. Regardless of the topologies and the number of nodes, Section 6.2.2 illustrated how the properties still hold. This chapter also shows the abilities of bigraphs and simulation to analysis RPL protocol on the same set of topologies (Section 6.3). More topologies can be examined if needed using our approach and tooling. However, 100 topologies were sufficient to draw our conclusion that the best approach for verifying the routing protocols is to combine both a simulation, as a quick proof for the decision-makers, and formal modelling and verification for comprehensive verification results. The chapter also demonstrated that, unlike simulation, bigraphs act as a diagrammatic notion that provides a clear and precise description for protocols (Section 6.4).

Our initial model (Section 5.1) and the comparison shown in this chapter, do not consider the timing aspects of RPL. However, we improve the model by showing that it is possible to encode digital clocks by leveraging the Markov decision processes semantics provided by *action bigraphs* [9]. This is shown in the next chapter, where an extended RPL timed model is presented.

Chapter 7

Real-Time Systems Modelling with Bigraphs

Utilising various bigraphs extensions, bigraphs have been successfully used in the literature to model a variety systems including mixed-reality games [16], cloud systems [123], self-adaptive fog systems [124], sensor network infrastructure [129] and rational agents [11]. One extension that has not been explored is real-time bigraphs that can model systems exhibiting non-deterministic behaviour that is controlled by time constraints. The non-deterministic aspects of real-time systems could be modelled by action bigraphical reactive systems (ABRSs), in which the underlying Markov decision process (MDP) semantics allows for a transition choice at each state.

However, there is currently no notion of *clocks constraints* in the bigraphs theory, *e.g.*, specifying that, after 2 time units have passed, a given action *must* occur. The state-of-the-art of bigraphs, which have been successfully extended, fosters our belief that bigraphs may model real-time systems in general, including protocols timing aspects. Therefore, this thesis proposes a modelling strategy to encode timed systems within ABRSs through the well-known MDP-based digital clock approximation.

This chapter proposes a modelling strategy to express clocks within action bigraphs (Section 7.1). It describes encoding clocks in practice using BigraphER in Section 7.1.1, while Section 7.1.2 elaborates on the set of reaction rules to model clock constraints in real-time systems through a digital clocks approximation. Then, the chapter shows the proposed technique in practice by modelling examples, *i.e.*, PTA and cloud system requests, alongside the model analysis in Section 7.2. The routing protocol for low-power and lossy networks (RPL) is extended in Section 7.3 to include some timed aspects that we abstracted in Chapter 5. Then, the chapter discusses the approach of extending bigraphs with clocks in Section 7.4. The chapter also includes a brief review of various modelling formalisms that have been extended to include clocks and other works that model timing aspects in bigraphs (Section 7.5). It offers concluding remarks in Section 7.6.

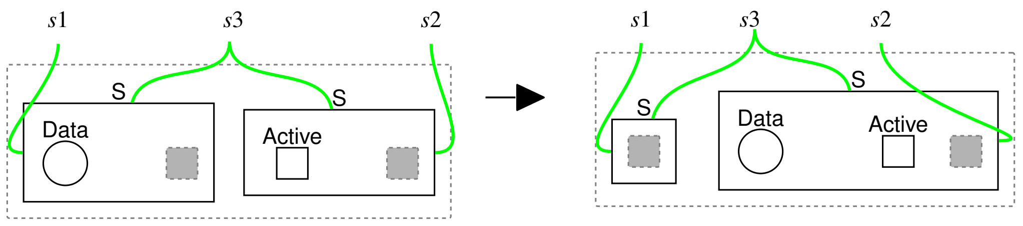


Figure 7.1: Reaction rule `send_Data` applies when the receiver is active.

7.1 Clocks Models in Bigraphs

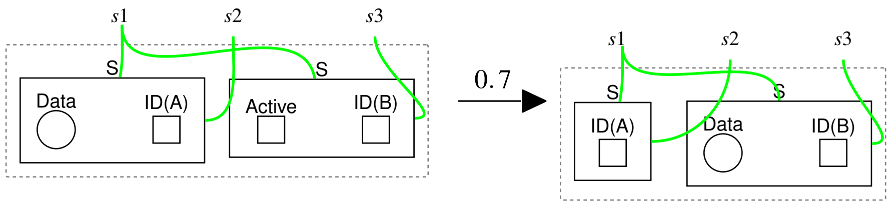
The big idea is that, by utilising action bigraphs [9], we can encode timed systems. This is based on a well-known approximation of (probabilistic) timed automata for Markov decision processes [54]. Intuitively, we extend the usual action semantics to allow two forms of action: *discrete actions* that encode system events (which may only be enabled at some time) and *time* actions that progress time.

Action bigraphical reactive systems (ABRS) allow for a choice of probability distributions at each rewriting step by assigning *weights* to the reaction rules, and by giving action labels to specific sets of reaction rules. An action is enabled/possible whenever there is a reaction rule from the set that is enabled. The resulting transition system is a Markov decision process (MDP) [15, 58], which can be verified using off-the-shelf model checkers such as PRISM [77].

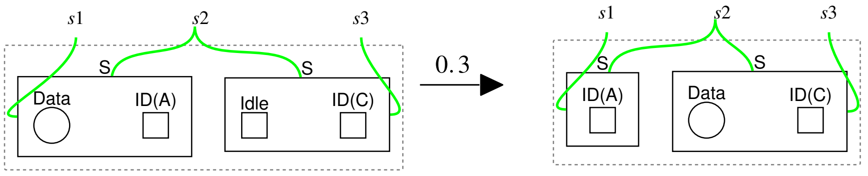
To show how ABRSs model non-determinism behaviour of real-time systems, where the underlying transition system is an MDP, we use the bigraph example shown in Figure 7.1 to include a choice of probability distributions. As an example, we can permit sensor `A` to send Data to either sensor by replacing the `send_Data` rule (Figure 7.1) with two rules, each representing a non-deterministic action through assigning the related rule with a weight (Figure 7.2). For example, with $weight = 0.7$, sensor `A` sends Data to sensor `B`, or to `C` with $weight = 0.3$. Weights are then normalised to probabilities depending on which rule(s) are applicable in a state. We use *bigraphs* throughout this chapter to mean *action bigraphical reactive systems (ABRSs)*. Using BigraphER [127], the only tool supporting ABRSs, we export the corresponding MDPs transition system to be formally checked. Importantly, action bigraphs still respect any rule priorities. This means an action will fire with any high priority rule before firing with those of lower priorities. We use this feature in our encoding of clock invariants.

Probabilistic timed automata (PTA) introduces support for clocks. As seen in (Section 4.1.3), transitions are associated to actions and probability distributions, with the addition of *clock constraints* (*e.g.*, $x > 3$) and *clock resets* (*e.g.*, $x := 0$). In the *digital clocks approximation*, we present the semantics of PTAs as a Markov decision process where states (including the initial one) are all the locations and clock invariants are met. We form a single action set consisting of both actions that manipulate time and user-specified discrete actions as follows:

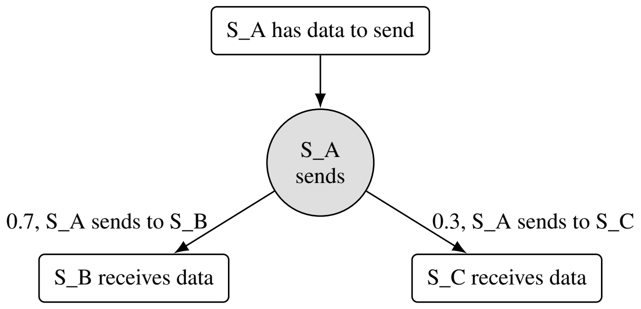
- a *discrete* action is enabled if all clock constraints (given in E) associated with the transition



(a) $Sensor_A$ sends to $Sensor_B$ with weight 0.7



(b) $Sensor_A$ sends to $Sensor_C$ with weight 0.3



(c) Non-determinism model with ABRSs

Figure 7.2: Example action bigraphical reactive system: probabilistic reaction rules using weights.

are satisfied;

- a *time* action is available while invariants associated with $l \in L$ are satisfied as time elapses. That is, we cannot progress time if something must occur beforehand.

Since we use action bigraphs, we can associate discrete transitions with a probability distribution. If required we can, for example, draw them with uniform probability to recover semantics for non-probabilistic timed automata.

7.1.1 Encoding Clocks in BigraphER

Standard BRSs evolve to a new state whenever there is a match of a reaction rule[1]. In this sense, they are non-deterministic but not explicitly so, *i.e.*, we cannot label particular actions without ABRSs. Models of real-time systems need to find enabled matches *and* meet any requirements introduced by clock constraints.

We propose a modelling technique that encodes digital clocks as *entities* to model real-time systems. Although PTAs can operate with real-valued clocks, for the digital clocks approximation,

[1]Subject to any requirements about rule priorities and conditions.

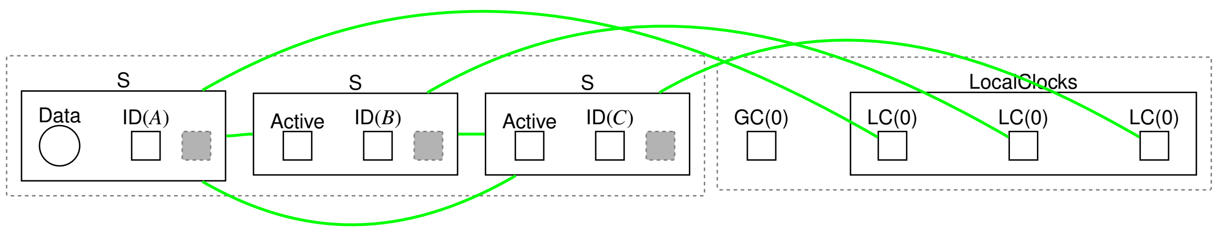


Figure 7.3: Initial state of example in Figure 7.1, but extended using a clock perspective.

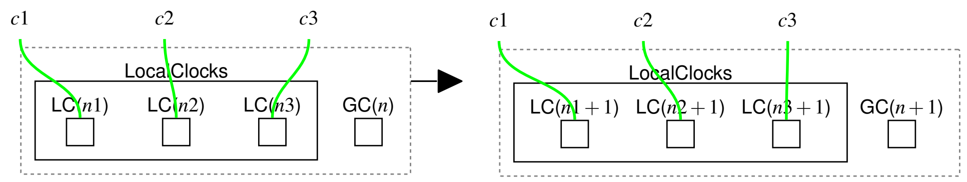


Figure 7.4: Reaction rule `clock_advance`$(n1,n2,n3,n)$ for a three timed entity system.

we fix clock values to non-negative integers and bound the total runtime of the system (to ensure a finite action set, which means the models can be analysed through existing MDP reachability techniques). As for the standard digital clock approximation, our approach does not support strict inequalities and comparisons between clocks. Since we use action bigraphs, these clocks can reside within the bigraph model itself, and a separate type of semantics is not needed (unlike for probabilistic bigraphs, for example).

We present our approach via an example; the main idea is given in Figure 7.3. We introduce a clock, as a new bigraph entity, for each timed entity in the system. Then, we position these clocks in a separate region so that all the clocks can be manipulated without polluting the actual system model. We place all the clocks into their own parallel region, which we call the *clocks perspective*. A global clock is represented by an entity family $\mathsf{GC}(n)$ – *i.e.*, there is one entity for each $n \in \mathbb{M}$ for some max time $\mathbb{M}$ – and it is used to model *wall-clock time*. Wall-time cannot be reset.

Local clocks are entity families $\mathsf{LC}(n)$, where $0 \leq n \leq \mathbb{M}$ for each timed entity. For clarity of modelling, we place these in LocalClocks, although this is not strictly required. We identify a set of *timed* entities and expand their arity by 1 to link them to a specific local clock. Using this method, we can *identify* a clock through the linked entity, *i.e.*, clocks do not need specific identifiers of their own. Not all of the entities in the system will own a clock, *e.g.*, Data is not timed. We assume all the clocks are initialised to 0 in the initial state.

The approach to position clocks within their own region is a design choice but, because of the flexibility of bigraphs, other choices are possible. For example, we could *nest* local clocks within particular timed entities instead of linking to them. We chose the extra region since it does not clutter any existing model, *i.e.*, we can convert an existing model to a timed representation without significant changes (other than arity) within the existing regions.

7.1.2 Reaction Rules for Digital Clocks

In our digital clock approximation, we have two main types of action: *discrete* actions, which are user-defined and system specific, and *time* actions that deal with the passage of time. As actions in ABRSs are modelled as sets of reaction rules, the main challenge here is defining appropriate reactions for each type of action.

For timing actions, the main rule is `clock_advance(x,y,...)`, as shown in Fig. 7.4. This rule advances all local and global clocks simultaneously. All clocks advance at the same speed (one unit per application). This is a parametrised rule that, like parametrised entities, defines a family of rules, *i.e.*, one for each possible value of the parameters. The *tick* action consists of the set of all possible `clock_advance(x,y,...)` rules for parameters drawn from $x \in \mathbb{M}, y \in \mathbb{M}, \ldots$, for some max time $\mathbb{M}$. Due to the parameter, only one instance of `clock_advance` will ever be enabled, meaning that this action always advances time with a probability of one.

For *discrete* actions, we extend existing system rules whenever a time constraint must be met. We take rules that have a standard (untimed) definition of a bigraphs rule ($L \rightarrowtriangle R$) and, similar to the clocks perspective, utilise parallel regions to link timed entities with their related clocks as follows:

$$\mathsf{L}_c \| \mathsf{LC}(n)_c \rightarrowtriangle \mathsf{R}_c \| \mathsf{LC}'(n)_c$$

Here, the notation $\|$ is the algebraic operator placing L and the new clock LC in *different* regions (as seen graphically by the dashed lines). The subscript c means that there is (at least) a link c that connects to other c linked entities. This is the new link that we use to identify specific clocks.

Importantly, this change also makes existing rules into parameterised rules (over parameter n). This is later used to encode timing constraints, *e.g.*, we chose a set of parameters that define at what (local) *time* a particular rule can be applied. For rules that are already parameterised, we can simply add an additional parameter for the clock.

As an example, we extend our rule in Figure 7.1, which allows sensors to send data, to only fire when the receiver is active *and* at least two time units have passed, *e.g.*, $\mathsf{LC}(n) \geq 2$. As we are modelling an inequality, we cannot do this with a single reaction rule. Instead, we provide a new parameterised rule `sending_data(n)`, as shown in Figure 7.5, that explicitly links the sending sensor to its local clock. The rule is only valid for $n \in \{2,3,\ldots,\mathbb{M}\}$, so this is the family of rules we generate. In this case, we include a clock reset on the right-hand side of the rule, and clocks may only be reset during the application of a rule.

Note that the `clock_advance` rule possesses the same priority as the corresponding rules, whose applications are triggered by clock constraints. This allows time to pass until the clock valuation satisfies the first invariant. In a such state, the non-deterministic choices are applicable, *i.e.*, the system can take a discrete action or permit the time to pass if it is not the last valid clock valuation in which an action must occur.

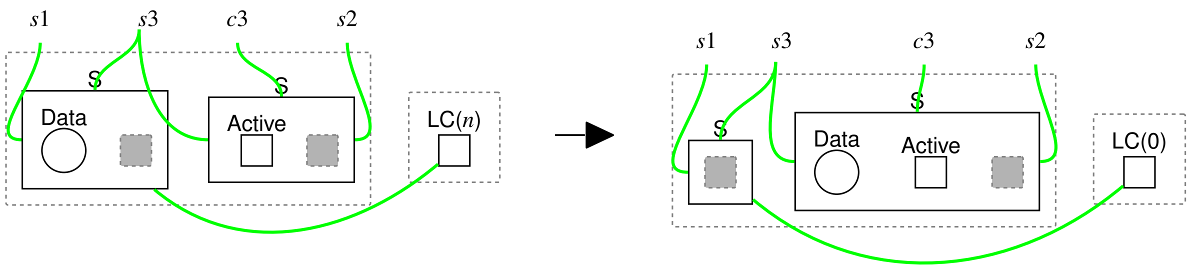


Figure 7.5: Reaction rule(s) `sending_data(n)`. Time constraints are encoded by setting $n \in \{2,3,\ldots,\mathbb{M}\}$.

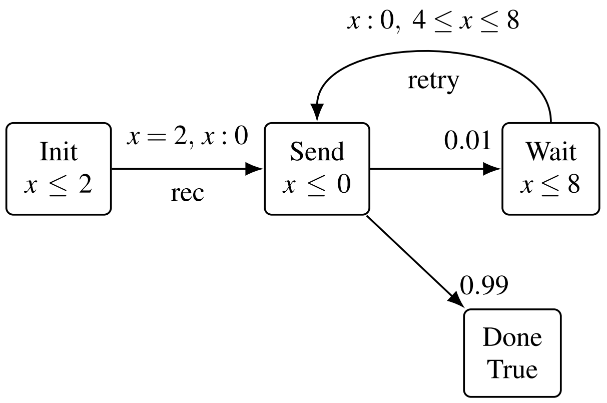


Figure 7.6: A probabilistic timed automata example model for a simple sending data process from ref. [78].

We must also encode any location-based clock invariants, *e.g.*, locations that are only valid for $x \leq 2$. In practice, this is used to force a transition to occur rather than allowing an indefinite wait within a particular location, and is almost always *true* (allowing indefinite waits) or a $\leq$ constraint. To encode these invariants, we make use of priority classes in the ABRS. For a state invariant, *e.g.*, $t \leq x \leq n$, we add any outgoing transition rules `r(n)` such that $\{$`clock_advance(...),r(t),r(t+1),...,r(n-1)`$\} < \{$`r(n)`$\}$. This means `r(n)` applies *before* any clock updates and *forces* a state transition before the invariant is broken.

7.2 Examples and Implementation

We implement our approach in BigraphER, which also generates the MDPs (which are the semantics for our digital clocks representation). Exported MDPs can be formally checked using standard (probabilistic) model checkers, *e.g.*, PRISM, allowing us to benefit from existing model-checking algorithms.

Modelling a timed system with our approach follows these general steps:

- Define sets of clock valuations/invariants according to the system requirements.

- Define a new clock perspective and add clock entities equal to the number of timed entities required, and one global clock. Increase the arity of the timed entities by 1 and link to the local clock. All clocks are initialised to 0.

- For a given $\mathbb{M}$ (max time) and a number of timed entities, generate the set of

  $$\texttt{clock\_advance}(\{0,\ldots,\mathbb{M}\},\{0,\ldots,\mathbb{M}\},\ldots)$$

  rules.

- Extend system rules by adding in the clock perspective (when required) to restrict their application based on the time constraints. The parameters of these rules must be in accordance with the clock invariants. Any state-based invariants are encoded using priorities.

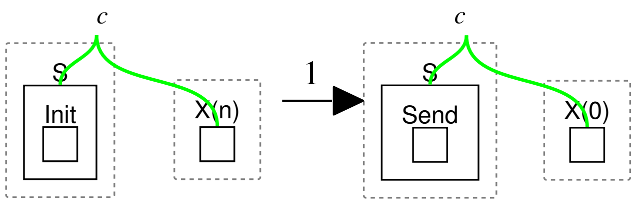
We illustrate our approach by encoding two different examples: a probabilistic timed automata example [78] and a cloud system request allocation [82].

7.2.1 PTA Example

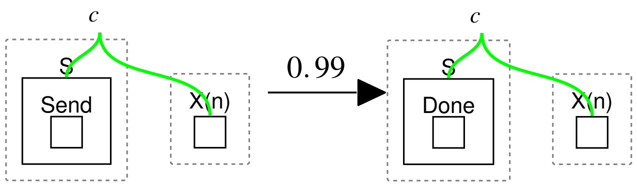
We apply our approach to the probabilistic timed automata example shown in Figure 7.6 and recreated from ref. [78]. This simple communication system attempts to send data based on the clock X constraints. Here, we do not model the actual sends, only the state machine the system goes through. This is somewhat unnatural for bigraphs, where we are more concerned with global system updates (possibly of multiple agents) than encoding a particular state machine for an entity, but it is used to illustrate that we can recover existing PTA semantics.

The system starts in the initial state `Init`. When the time elapsed is exactly 2 time units (as forced by the transition constraint *and* the clock invariant in the state), the system moves to the `Send` state. The clock invariant $n \leq 0$ forces the data to be sent immediately. With a probability of 0.99, the system reaches its final state, `Done`. Alternatively, with a probability of 0.01, it fails and moves to a `Wait` state. Here, the system waits at least 4 time units and at most 8 time units (forced by the invariants) before moving back to the `Send` state to retry. The clock is reset with *two* transitions: from `Init` and `Wait` states.

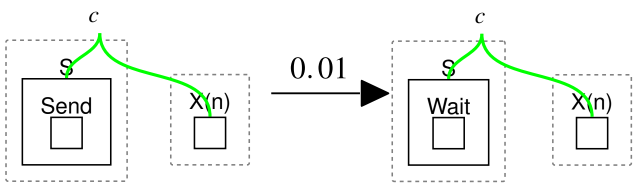
We begin modelling this PTA example by defining the required time constraints. For each state, we define the valid clock valuations as follows. The `init_transition` rule (Figure 7.7a) applies over $n \in \{0,1,2\}$ for the `Init` state, and the rules that are associated with the `Send` state (Figure 7.7b and Figure 7.7c) fire immediately upon receiving data, *i.e.*, at $n = 0$. While the `wait_transition` rule (Figure 7.7d) is applicable over $n \in \{4,5,6,7,8\}$. For all the parameters, except 2 for `Init` and 8 for `Wait` states, the rules application should be in the same priority class as the `clock_advance(...)` rule enabling the non-deterministic behaviour: time passes or data sends. However, as the invariants force leaving if more than $\mathbb{M}$ units elapse, `init_transition(2)`, `send_transition(0)` and `wait_transition(8)` are in a higher priority class so they apply once the $\mathbb{M}$ is satisfied. We encode a single rule for each



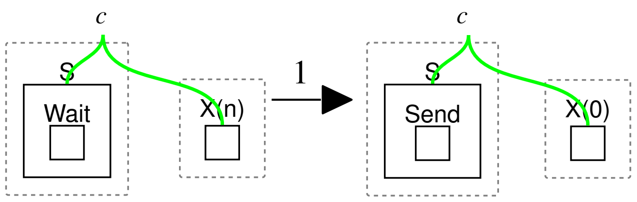
(a) Reaction rule `init_transition(n)`, $n \in \{0,1,2\}$



(b) Reaction rule `send_transition_success(n)`, $n = 0$



(c) Reaction rule `send_transition_fail(n)`, $n = 0$



(d) Reaction rule `wait_transition(n)`, $n \in \{4,5,6,7,8\}$

Figure 7.7: Reaction rules for the PTA example.

system transition and show how the probabilities are encoded using rule weights[2], *e.g.*, to allow the `Send` state to move to either `Wait` or `Done`.

In Figure 7.8, we show that the clock bigraph model is in accordance with the probabilistic timed automata example by giving the transition system. For space, we only provide the first few transitions. Here, the system moves from the initial state `Init` to the `Send` state. When the system is in the `Init` state and the clock constraints are satisfied, there exist two possibilities: time passes or action occurs. Once the clock becomes $X(2)$, the system *must* move to `Send`.

[2]In this case, as there is only one agent moving state, the weights are equivalent to their probabilities since there will never be additional rule application matches to account for.

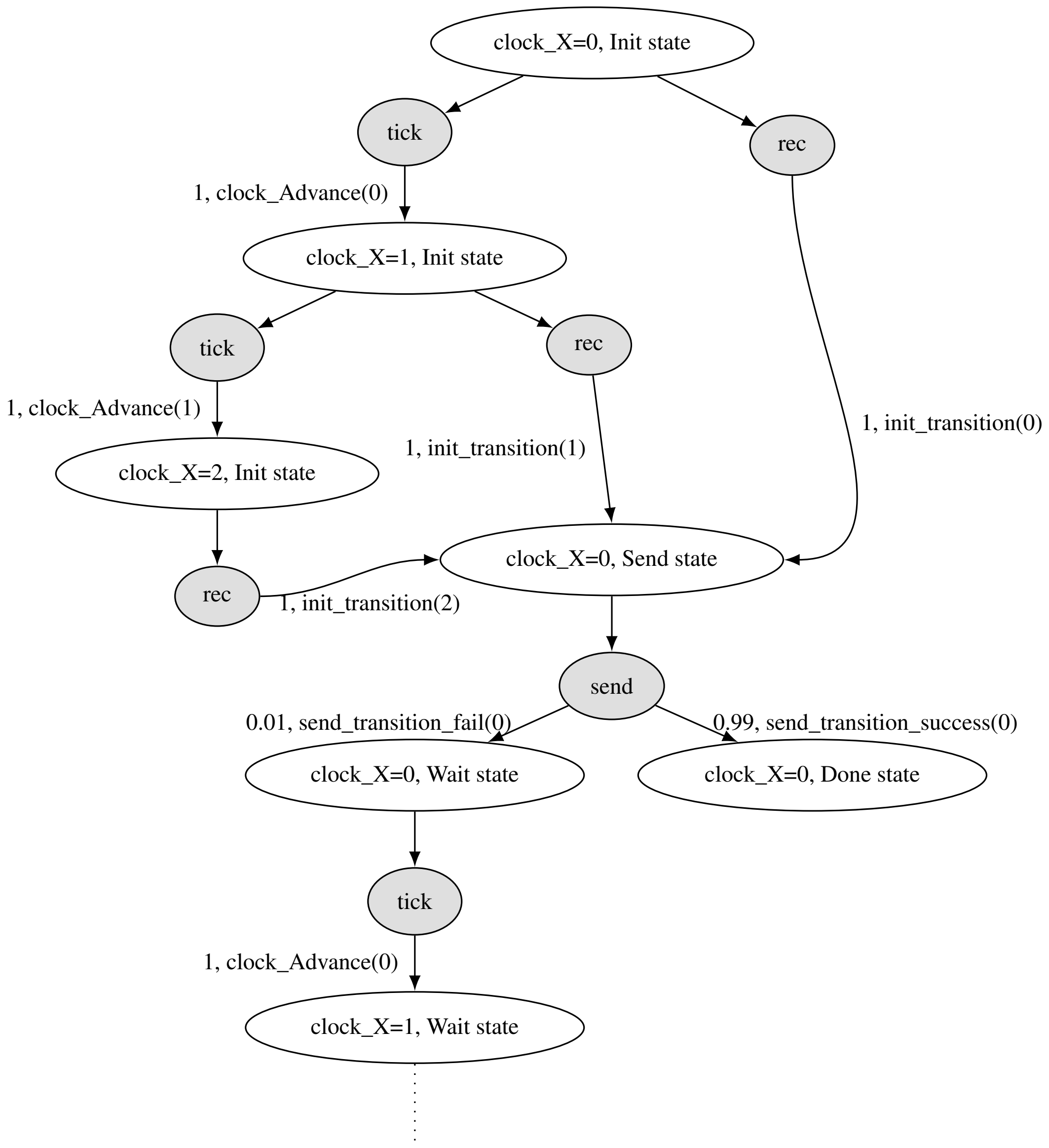
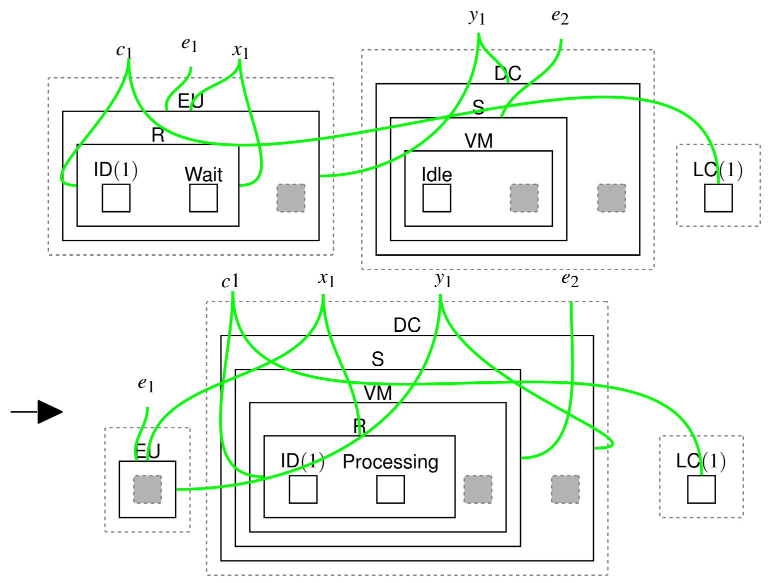


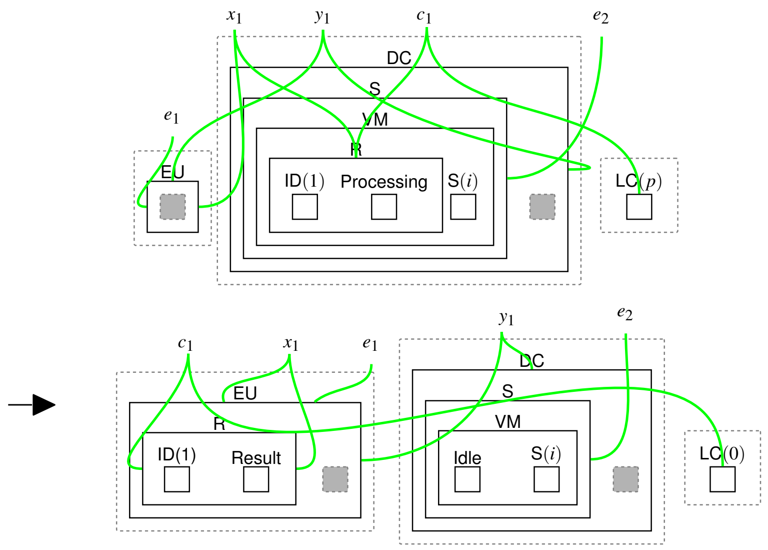
Figure 7.8: Resulting MDP (partial) for the probabilistic timed automata shown in Figure 7.6.

7.2.2 Cloud System Requests

A cloud system is a collection of resources that include hardware, software, networks, . . . etc. that is used to store, manage and process data. That is, end users (EU) send their requests (R) over the Internet to virtual machines (VM) hosted on physical servers ($\mathsf{S}(i)$ with $i > 0$) within a data centre (DC) for processing. Standard bigraphs can model and analyse the structure and dynamic behaviour of cloud systems, but not the time constraints that cloud applications usually have, *e.g.*, the duration of tasks. Time constructs are added to the BRSs to model and analyse cloud systems with time constraints in [82], but this work differs in that it expresses clocks as nested entities and needs multiple reaction rules to advance clocks (which can cause unnecessary state-space explosion). They do not have a method to track approach wall-time and, as they do not use action BRSs, cannot model the (explicit action) non-deterministic behaviour of many real-time systems.



(a) Reaction rule `sendRequest(s)`, $s = 1$.



(b) Reaction rule `processRequest`(p,i), $p = 3$ when $i = 1$ and $p = 4$ when $i = 2$.

Figure 7.9: Processing rules for the *first* request.

The example in that work models a cloud system that has *two* end users connected to each other through a link, *e.g.*, e_1; and *two* servers that are also connected via a link, *e.g.*, e_2. A data centre and an end user are related through a link, *e.g.*, y_1. There are four requests each of them connects to its end user through a link *e.g.* x_1. The requests need to be processed using two servers.

Analogous to this example, we model the example as follows. We give each request a parameterised identifier ($\mathsf{ID}(r)$, where $r \in 1,2,3,4$). We also associate each request with different controls to reflect its current status. Initially, all requests have the status Wait. Afterwards, requests that are being processed have the status Processing, while the status Result indicates that a request has been processed and returned to the end user.

A clock for each request is defined but, unlike in [82], a global clock that shows the wall-time is added. We also use a reaction rule that simultaneously advances all the clocks. For request processing, we specify one reaction rule, *i.e.*, `sendRequest` (Figure 7.9a), to send a request from an end user to a virtual machine for processing. Given that the servers have varying resources and, thus, require different amounts of time to process a request, two rules are applied to return the requests after processing, *i.e.*, once a predefined time has elapsed (Figure 7.9b). The rule `processRequest_S1` is for $\mathsf{S}(1)$ that takes 2 time units, and the rule `processRequest_S2` is for $\mathsf{S}(2)$ that needs 3 time units to process a request. We define the same time constraint assumptions stated in [82] as integer sets. These sets are to be used with the timed reaction rules. That is, for each request, there are four integer sets as follows. The *first* one is the valid clock valuation for a request. The *second* contains the time at which a request can start. The *last* one is two different sets that determine the time that a request should last according to the processing time for each server. We provide the four integer sets as follows:

- $N = \{n \mid 0 \leq n \leq \mathbb{M}\}$ for each request,
- $S = \{1, 3, 4, 5\}$ for requests 1, 2, 3, and 4 respectively,
- $P_1 = \{s + 2 \mid s \in S\}$ for requests processed on server $\mathsf{S}(1)$,
- $P_2 = \{s + 3 \mid s \in S\}$ for requests processed on server $\mathsf{S}(2)$.

The work in [82] considers virtual machine migrations. However, we allocate the process to a machine that is in the Idle status, indicating that it is free and ready to receive a request. Furthermore, we will show how our approach can be used to verify the interesting properties in the following section. We make the assumption that a request is always forced to send no more than 1 time unit *after* the deadline. This means processing time for delayed tasks can be faster than expected, *e.g.* $p_1 = s + 2 - 1$. This could be accounted for using additional entities, *e.g.* Delayed, or out of time requests could be dropped. Bigraphs give the flexibility to experiment with these approaches with additional rules.

7.2.3 Real-Time Model Analysis

To check the feasibility of our proposed modelling technique, we automatically export the bigraph models to MDPs using BigraphER. We can then verify them against the required properties using PRISM by expressing the properties in probabilistic temporal logic (PCTL) [50]. To help write properties, we use *bigraph patterns* that allow states to be labelled based on matches [16], *e.g.*, label any state that has a Data entity. Since clocks are just part of the model, we can also label clock values, *e.g.*, $clock\_LC_0$ for $\mathsf{LC(0)}$. Interestingly, the clocks used in the properties are those employed in the model, *i.e.*, they are just bigraph entities. This contrasts with clock variables that would be generated if, for example, the PTA was expressed directly in PRISM. This provides flexibility, *i.e.*, we can write predicates for many different types of clock matches.

We make use of the following PCTL quantifiers to specify our properties: **A** (for all) and **E** (there exists), and path formulae, **F** (eventually), **X** (next), **U** (until) and $\wedge$ (and). We also use the probability quantifier **P** to check the probability of reaching a specific state. For example, $\mathbf{P} \leq 0.5\,[\mathbf{F}\, stateA]$ means eventually there is, at most, a probability of 0.5 that the system reaches a state satisfying *stateA*, we label a state as *stateA* using a bigraph pattern.

We conduct model checking on several interesting properties. For example, given the work presented in Section 7.2.1, we can perform probabilistic reachability properties such as “Is there at least a 0.99 percent probability that the system moves to the `Done` state successfully?”, which holds for this system.

$$\mathbf{P} \geq 0.99\,[\mathbf{F}\,(\mathtt{In\_Done\_state})]. \tag{7.1}$$

We can also utilise clock predicates to check properties relating to time, *e.g.*, “eventually there is at least a 99% chance that the system moves to the `Done` state and resets clock X”, which is true for our system, *i.e.*,

$$\mathbf{P} \geq 0.99\,[\mathbf{F}\,(\,\mathtt{In\_Done\_state} \wedge \mathtt{clock\_X}_0)]. \tag{7.2}$$

Importantly, these are based on our clock entities, not time-bounded properties within PRISM, which operate on their own time units.

We can also check that the clock invariants are satisfied, *e.g.*, that the system is not in the `Wait` state after 8 time units have elapsed:

$$\mathbf{A}\,[\neg(\mathbf{F}\,(\mathtt{In\_Wait\_state}) \wedge (\mathtt{clock\_X}_9))]. \tag{7.3}$$

As expected, this is true in all cases.

For the cloud system we can, for example, check the first request is always sent once *one* time unit has elapsed. As we use next (**X**), the request must be sent immediately before any other state transition.

$$\mathbf{A}\,[\mathbf{F}\;\mathtt{clock}_1 \wedge (\mathbf{X}\;\mathtt{request}_1\mathtt{\_sent\_to\_S})] \tag{7.4}$$

7.3 RPL Model Revisit

In Chapter 5, we demonstrated how bigraphs model routing protocols and successfully find all the valid routes. However, since bigraphs have no notion of *clocks constraints* in theory, we did not consider its timing aspects, which has no effect on finding all the valid routes. However, since we have now introduced the clock notion into bigraphs, we revisit the RPL initial model to include some of the RPL timing behaviour. Specifically, we model the DIS sending process.

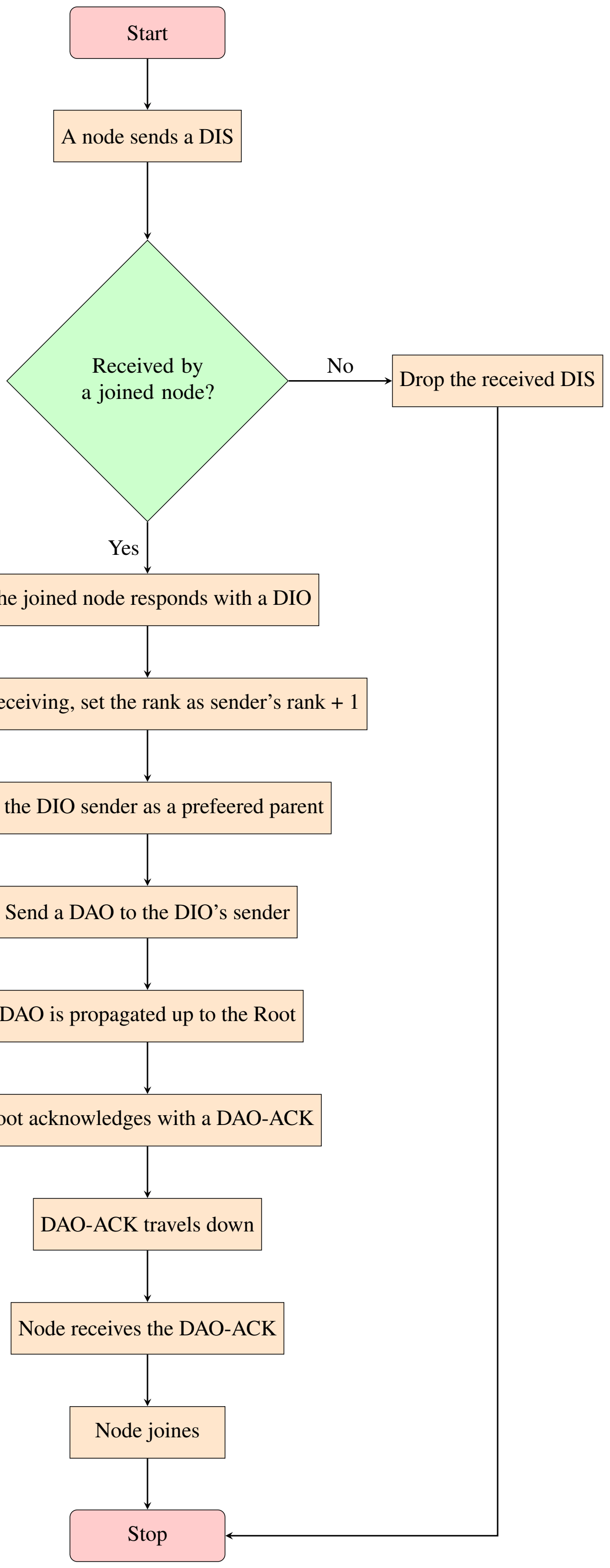

Start
A node sends a DIS
Received by a joined node?
No
Drop the received DIS
Yes
The joined node responds with a DIO
Upon receiving, set the rank as sender's rank + 1
Set the DIO sender as a prefeered parent
Send a DAO to the DIO's sender
DAO is propagated up to the Root
Root acknowledges with a DAO-ACK
DAO-ACK travels down
Node receives the DAO-ACK
Node joines
Stop

Figure 7.10: Flow chart for the DAG construction based on a DIS message, for *one* generated DIS

To model the DAG construction process that is initiated by sending DIS messages, various changes and updates are needed. First, we introduce clocks into the initial RPL bigraphs model and then model the required timing aspects, *i.e.*, DIS messages. In the following, we demonstrate these changes by applying them to the model shown in (Section 5.1).

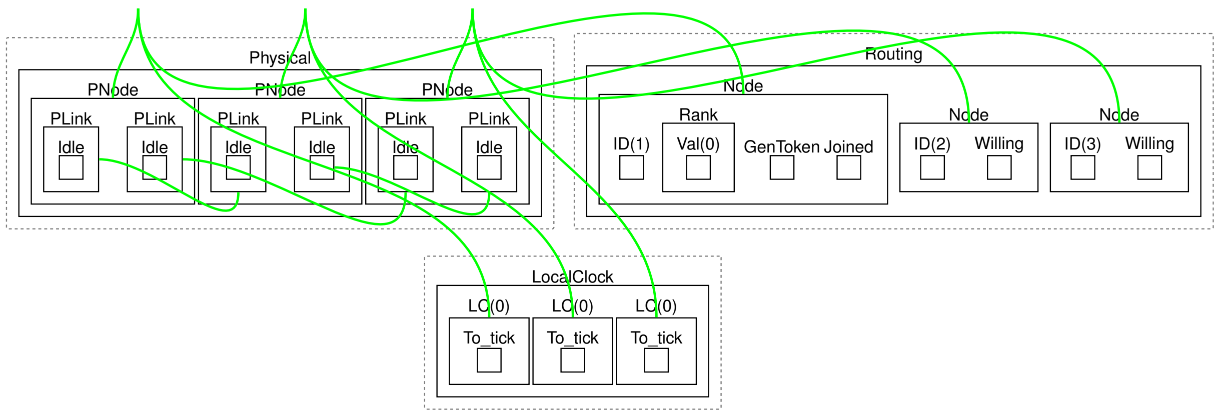


Figure 7.11: Bigraph representation for a WSN topology example.

7.3.1 Update the Topology

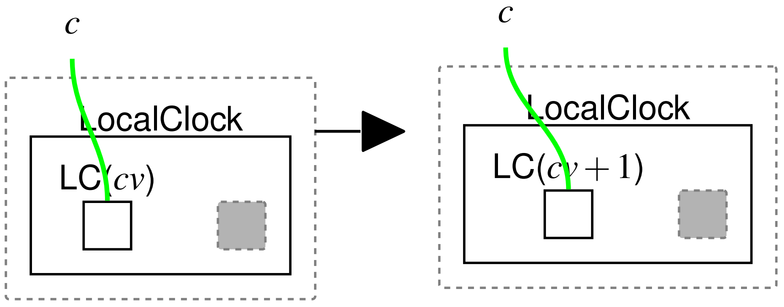
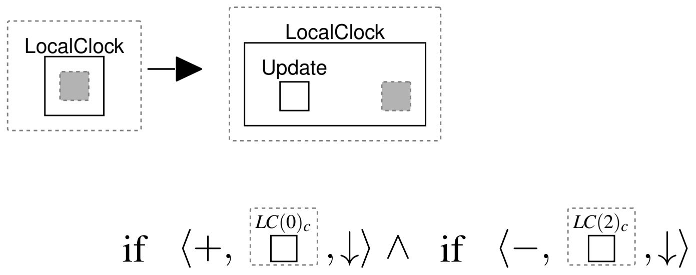


Figure 7.12: Reaction rule `updated_clock_advance(cv)`, $cv \in \{0,1,\ldots,\mathbb{M}-1\}$.

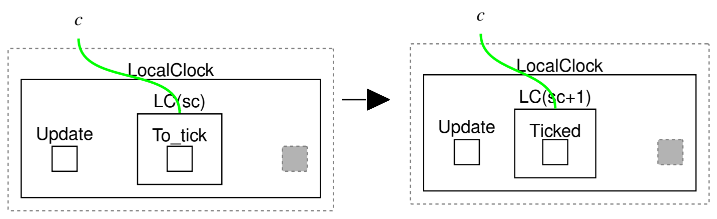
We specify a number of clocks (one for each node), including the root, in a separate region. As seen before, each clock is linked to its node via a link. We add a control To_tick, indicating that this clock is supposed to advance (more details are given soon). We also add a control Willing to each node, except the root, to indicate that a node is willing to join a nearby DAG. Additionally, we explicitly indicate that a root is joined by adding the Joined entity.

7.3.2 Adding Clock Advance Rules

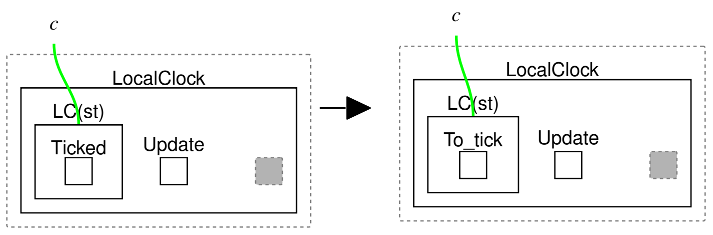
Applying the proposed technique to RPL creates some challenges due to the following. In the RPL, clocks advance synchronously but their associated actions succeed in various likelihoods. That is, a direct child of the root sends a DIS that is received by the root, while the grandchildren



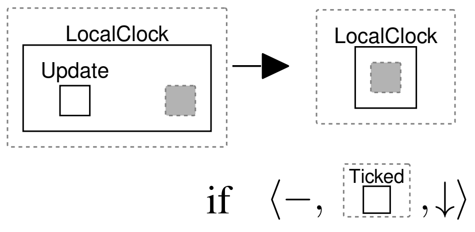
(a) `updated_clock_advance`



(b) `tick(sc)`, $sc \in \{0,1\}$



(c) `tickDone`, $st \in \{1,2\}$



(d) `updated_clock_advance_Done`

Figure 7.13: Reaction rules for clock advance.

send a DIS that is not received since there are no joined nodes yet in their range (as we will see soon). That is, more time is needed as we traverse further away from the root. Due to the fact that we require our model to operate on non-predefined topologies, we are not able to set different clock evaluations for different nodes. The other issue is that there exists a clock for each timed agent, as seen for `clock_advance` (Figure 7.4) that explicitly specifies the number of clocks. Since we have illustrated that our RPL model is applicable on a non-predefined topology, we generalise the `clock_advance` rule to be applicable on a non-predetermined number of nodes, and consequently clocks. To achieve this aim, we encode the following rules:

- We update the `clock_advance` rule to a `updated_clock_advance` that contains

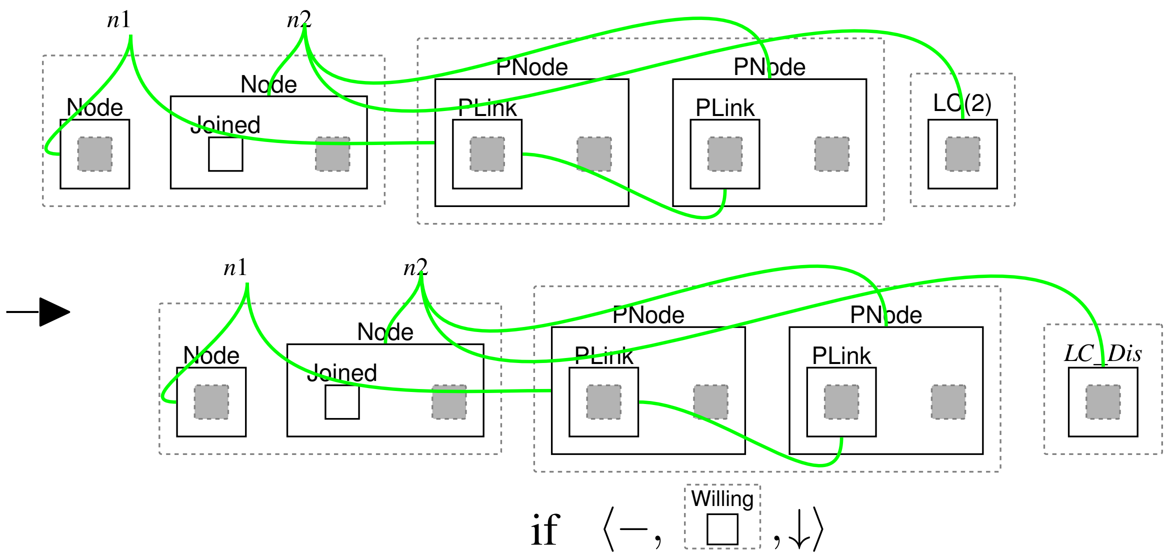
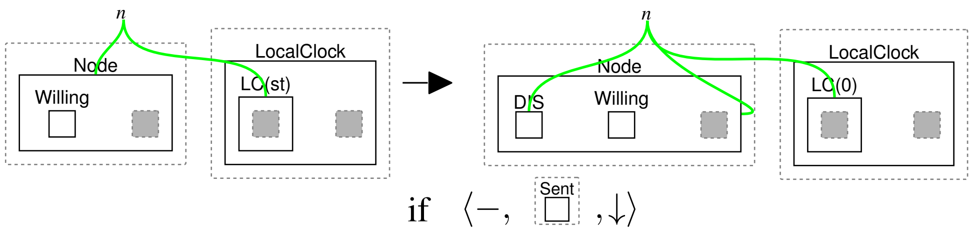


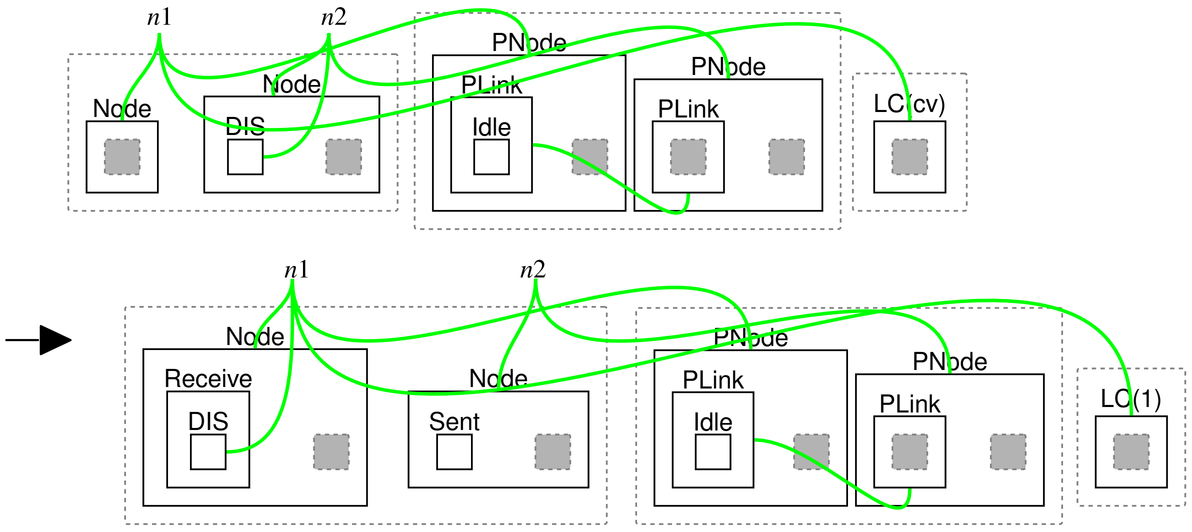
Figure 7.14: Reaction rule `clockDisable`.

only one clock, as seen in Figure 7.12. As a result, this rule applies once for each clock in the model, which leads to a growing state space. To minimise the state space, we make this rule an instantaneous rule (Section 4.4.3). However, this hides non-deterministic choices, *i.e.*, a discrete action or a time action, within the transition system. To tackle this issue, while maintaining a possible minimum state space, the other three reaction rules are added (which are described below).

- `updated_clock_advance` rule is updated as in Figure 7.13a that adds a control Update as a flag, each time there is a need for time to pass. This rule plays the role of the `updated_clock_advance` rule (in Figure 7.12), but it applies once to indicate that all the node clocks will simultaneously advance, so it does not increase the state space. This rule enables the visibility of the non-deterministic behaviour in the transition system. Note that, since it will be a match every time this rule applies, we restrict the rule application by the condition that *not* $LC(2)_c$ in *param*. That is, time advances unless there is a clock that reaches 2, which is the last time we assume that the node can send a DIS. Note, as we said before, that closer nodes will be joined before the more distant ones. Since there is no specific rule for resetting, the application of the `updated_clock_advance` rule will be disabled by the root's children nodes since they will continuously advance (reaching 2) while they have already joined, *i.e.*, no action allows resetting. To tackle this issue, we add a rule that changes this control into *LC_Dis* once their nodes joined and no more nodes are willing to join within their range, as we will see soon (Figure 7.14). However, this results in another issue, the `updated_clock_advance` rule infinitely applies since there will



(a) Reaction rule `generateDIS(st)`, $st \in \{1,2\}$.



(b) Reaction rule `sendDIS(cv)`, $cv \in \{0,1,2\}$.

Figure 7.15: Reaction rules for sending a DIS message.

not be $LC(2)_c$ in *param*.

- To address the latter issue, we update the `updated_clock_advance` rule again by adding a compound condition that $LC(0)_c$ in *param* but not $LC(2)_c$. We duplicate the rule with a compound condition $LC(1)_c$ in *param* but not $LC(2)_c$ (unshown here). This duplication is to cover all the possible combinations for all the clocks, *e.g.*, 0 and 0, 0 and 1, etc. The first condition, as illustrated, is to compound the clock evaluation and the second one is to prevent the infinite application of the rule, as it states that it should be a clock that needs to be advanced.

- `tick` (Figure 7.13b). This rule performs the real ticking for the clocks. It advances each clock by *one* time units and marks the clock with a Ticked control. This control is to allow the clock to advance only *one time unit* each time it applies.

- `tickDone` (Figure 7.13c). This rule is to reverse the clock to its initial state, allowing time to advance again when required. Ticked is converted back to Totick.

- `updated_clock_advance_Done` (Figure 7.13d). This rule removes the Update control when there is no Ticked in any clock, indicating that the first round of advances for all the clocks is completed.

Note that there is still no separate rule for clock reset; the resetting occurs within the corresponding action, *e.g.*, sending a DIS.

The *four* rules, given above, work for any number of nodes. We give the discrete action, *i.e.*, DIS generation, and the time action, *i.e.*, `updated_clock_advance`, the same priority, so explicit non-deterministic choices, *i.e.*, tick or send DIS, are shown in the transition system.

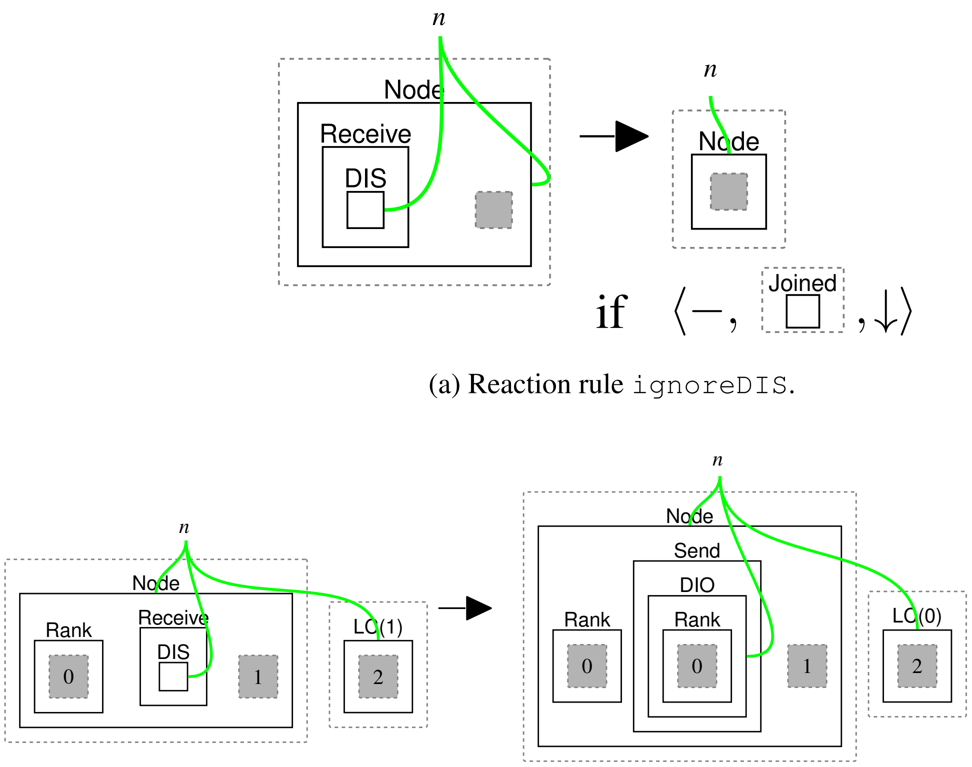
7.3.3 DAGs Construction Initiated by DISs

We assume that nodes periodically generate a DIS during each time unit specified as $sending\_Time \in \{1,2\}$, we also allow the *message delay*, *i.e.*, nodes can send on either time 1 or 2. The already joined nodes respond with a DIO, and then the joining process starts as is in the initial model. The following steps illustrate the encoding rules to achieve the DAG construction that is initiated by the DIS control message.

Sending a DIS message (Figure 7.15). We specify a timed rule, `generateDIS`, that applies only when the time constraints for the willing to join nodes are satisfied, *i.e.*, $st \in \{1,2\}$ (Figure 7.15a). The rule generates a DIS message while keeping the Willing control so that the node can try again if it does not succeed. A node preserves its Willing control until it receives a DAO-ACK message so joins a DAG. Note that `generateDIS` is applicable if there is no Sent in *param*, which is true in the first place so that allowing nodes to generate and send one DIS at a time. To maintain non-deterministic behaviour, both `updated_clock_advanc` and `generateDIS` are in the same priority class. The node then sends the DIS message to all of the nodes that are within range (whether they are already joined or not), and resets their clocks to the time that the DAG's nodes can send DIOs, *e.g.*, $n = 1$ to allow instant response (Figure 7.15b).

Handling a DIS message (Figure 7.16). Nodes, upon receiving the DIS, decide what to do according to their status. One of the *two* reaction rules apply:

- `ignoreDIS` conditional rule (Figure 7.16a). Applies when the receiver node is not part of an existing *DAG* – which we indicate by the absence of a Joined control. In this case, the node drops the received DIS.



(a) Reaction rule `ignoreDIS`.

(b) Reaction rule `receiveDIS`.

Figure 7.16: Reaction rules for handling a DIS message.

- `receiveDIS` rule (Figure 7.16b). Applies when the receiver node is part of an existing *DAG* – which we indicate by specifying a Rank control on the left-hand side. The receiver node generates a DIO message that contains the current rank of the node.

Replying with DIO (Figure 7.17). The receiver node, which is part of a DAG, sends a DIO to the node that sent the DIS to indicate its willingness to join. Upon receiving the DIO, the Sent control is dropped so that the node can generate another DIS if required later. As previously seen (in Chapter 5), the node that received a DIO replies with a DAO that is sent to the root. The root acknowledges the node that initiates the process with a DAO-ACK.

Joining the DAG (Figure 7.18). Here, we update the `toJoin` rule to indicate that a node receives a DAO-ACK and joins the DAG, it then drops the Willing control so no more DISs are generated.

Multicasting DIOs (Figure 7.19). A joined node multicasts DIOs to all the other joined nodes in its range. This enables these nodes to improve their ranks. That is, if a node receives a DIO from a neighbour node that gives it a better rank, it updates its current parent with that node and its rank accordingly, as illustrated in Section 5.1.3.

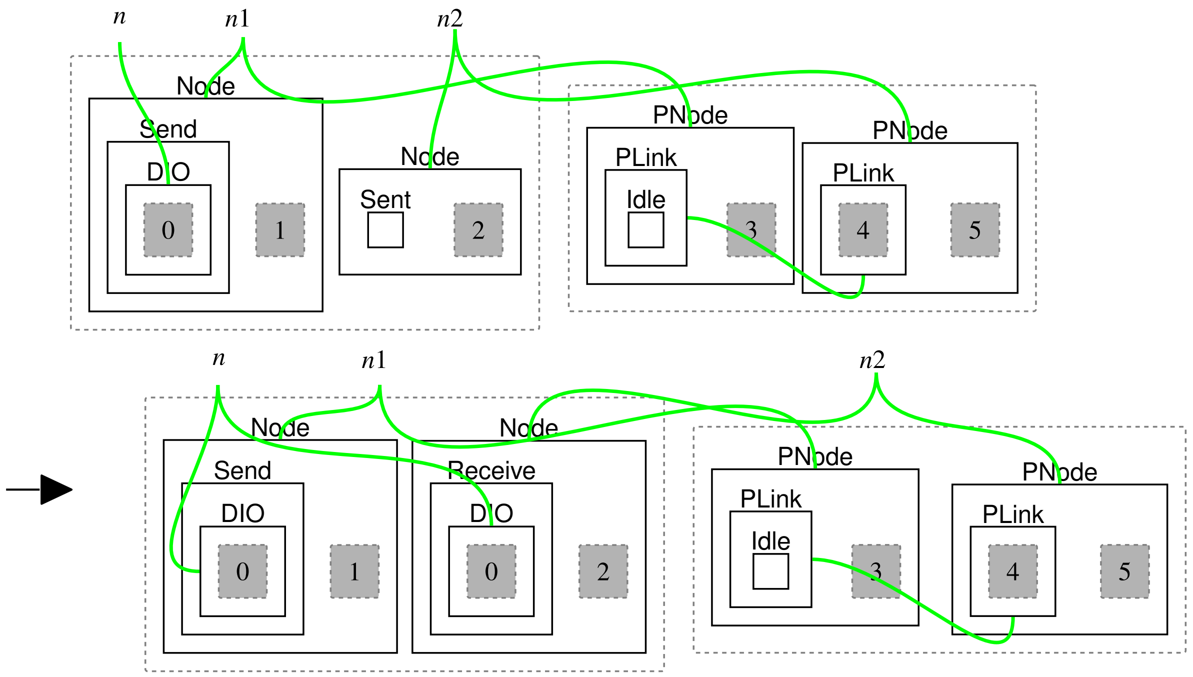


Figure 7.17: Reaction rule `sendDIO`.

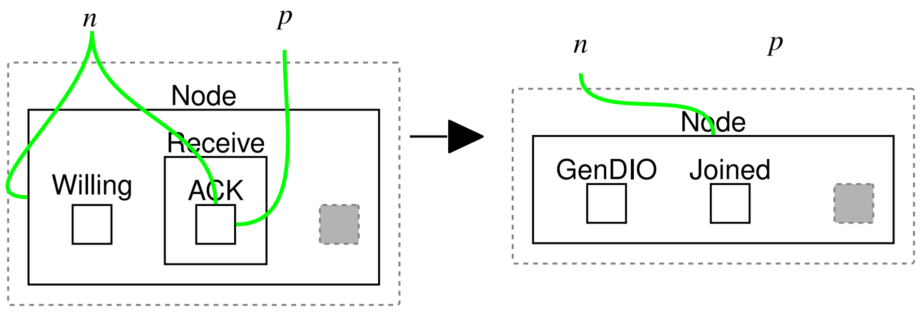


Figure 7.18: Reaction rule `toJoin`.

Failed DIS (Figure 7.20). We assume that a node that sends a DIS (*i.e.*, Sent exists), its clock reaches 1 again (*i.e.*, already tried to send DIS) and is still willing to join (*i.e.*, Willing exists) is considered to be a node that fails to successfully send a DIS (Figure 7.20a). We label this node with a Fail control and allow the nodes within its range to send a DIO to the failure node directly (`DIO_To_FailedNodes`). This terminates the process of continually sending DISs that are not received (Figure 7.20b), so allow all nodes to eventually join and valid DAGs are constructed.

Dropping extra controls. Since BigraphER merges identical states, *one* rule drops all clocks is encoded (not shown) since they might have different valuations for different possible pathes.

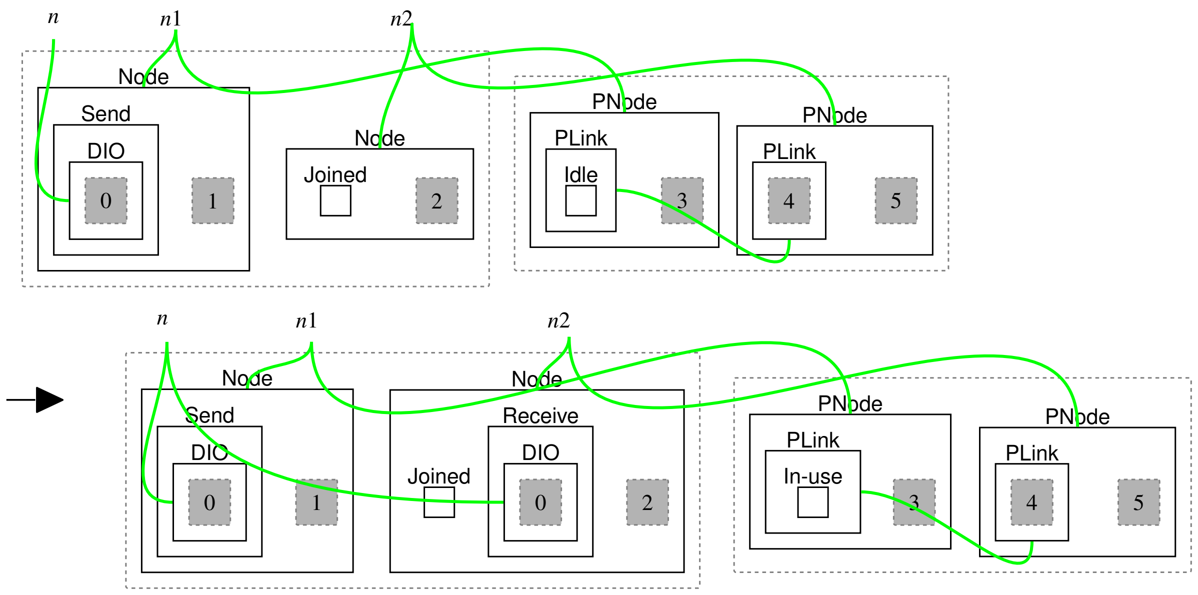


Figure 7.19: Reaction rule `multicastDIO`.

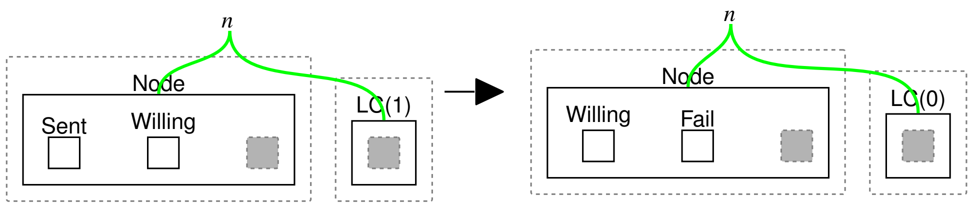
7.3.4 RPL Model Checking

We verify that our RPL timed model successfully satisfied the same properties as the ones checked for the initial model. Namely, all the nodes eventually join, they join with an optimal rank and the DAGs are cycle-free. Additionally, we can check the timed behaviour of RPL, as we demonstrated in Section 7.2.3. One example might be that a DIS message is periodically sent, *i.e.*, at LC(1), LC(2), ..., until it is received. That is, the property ensures the prohibition of intermediate states so that a time interval must be followed immediately by the transmission of a DIS message, *i.e.*,

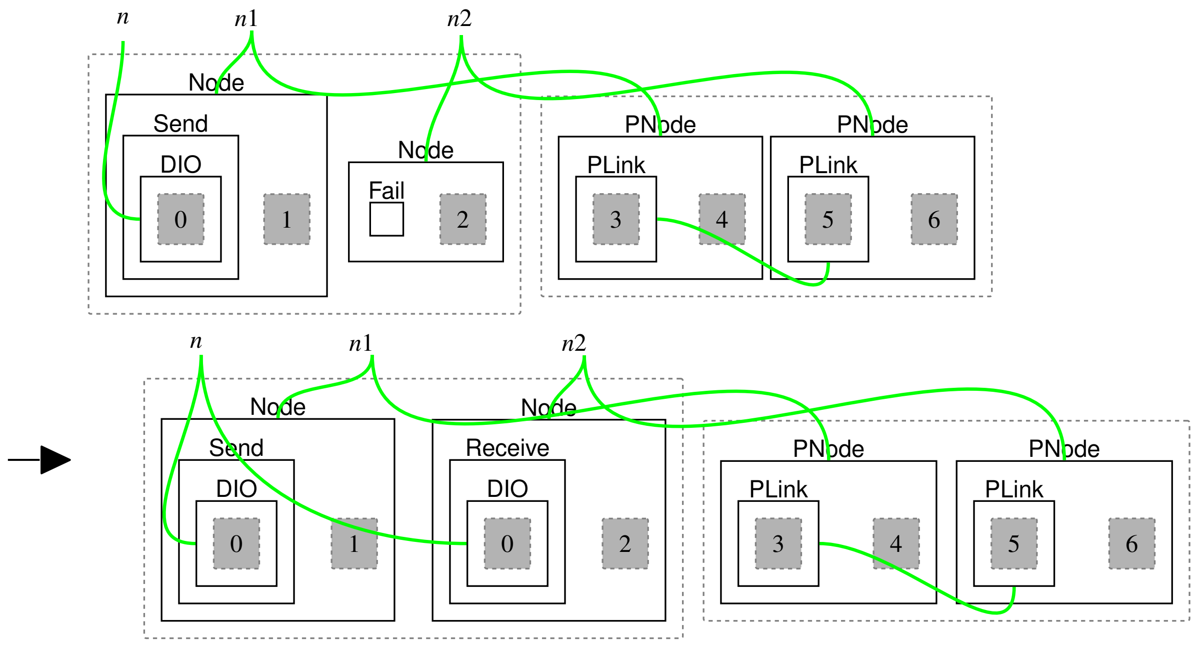
$$\mathbf{E}\,[\mathbf{F}\;\texttt{clock\_LC(n)} \wedge (\mathbf{X}\;\texttt{DIS\_Sent})\;\mathbf{U}\;(\texttt{DIS\_received})], \quad \text{where } 1 \leq n \leq \mathbb{M}. \tag{7.5}$$

7.4 Discussion

In the previous section, we show how we apply our proposed approach of utilising the bigraphs in modelling real-time systems on the existing untimed model, *i.e.*, the initial RPL model. That is, we build on the initial model that we described in Chapter 5. We also update the extension comparison table (Table 5.2) with the timed model, as shown in Table 7.1. The approach of introducing clocks into bigraphs substantiates its feasibility. We show that our approach is applicable to a variety of examples, including PTAs and various time aspects in both cloud systems and routing protocols. Consequently, we believe that our approach is applicable to a wider range of real-time systems. Additionally, our technique supports multiple clocks and the transition system obtained via rewriting faithfully encodes the digital-clock approximation of the



(a) Reaction rule `sendDIS_Fail`.



(b) Reaction rule `SendDIO_To_FailedNodes`.

Figure 7.20: Failed to join via node sending a DIS.

behaviour of a real-time system.

However, this approach suffers from the same limitations as the digital-clock approximation, as now described. Currently, we do not support diagonal clocks and strict inequalities. In addition, the overall size of the resulting MDP grows exponentially with the number of clock variables [51]. We mitigate the state space explosion by adopting the following two strategies: bounding clock valuations, thus finite transition systems are obtained, and allowing the base time unit to be specified in each model, effectively allowing clocks to advance by multiple ticks in one step. However, the dimensionality of the state space increases as more clock valuations are considered. Another limitation of our approach is that we assume that clocks are always synchronised, *i.e.*, they all progress at the same speed, which might not always be the case in scenarios such as wireless sensor networks and IoTs.

Although our approach of introducing clocks into bigraphs accurately models real-time behaviour, it leads to an exponential growth in the RPL model state space, which poses challenges

Table 7.1: Updated RPL model extensions.

Extension	Controls	Rules	Update
RPL main model			
Initial models	24	16	–
DAO-ACK model	+3	+7	2
Attacks			
Sinkhole attack	0	+1	–
Helloflood attack	+1	+1	–
Blackhole attack	0	+1	–
Probablistic			
Link failure	0	+1	–
Message receive failure	+1	+1	–
Time aspects			
Clocks	+9	+7	–
DIS model	+4	+8	2

for scalability and analysis. However, this might not be the case for other types of real-time systems. In other words, the substantial increase in the state space may be caused by other related behaviour as follows. RPL enables nodes to send a DIS to seek the joining of an existing DAG, only nodes in this DAG can respond with a DIO. This makes the nodes that are distant from the root perform several sending DIS attempts with their clocks also performing multiple resets and advances. The other possibly related cause is that the nodes join through the first node that responds to their sent DIS, which may not give them the optimal rank. To address this issue, we allow for repair steps in which all the nodes multicast DIOs, so the nodes have the opportunity to improve their ranks. When this is the case, all the other rules (related to the construction of the DAG) will be performed again. In addition, since distant nodes might send the DIS repeatedly but they are not subsequently received, we allow a direct DIO multicast to those nodes to ensure that all the nodes eventually join the DAG. The other possible cause is that we enable a message delay and, thus, nodes can send a DIS either when their clocks satisfy 1 or 2. As examples of how the timed model explodes the state space, we compare four topologies for the two models, *i.e.*, untimed and timed RPLs, in terms of the number of states, as shown in Table 7.2. We provide some possible solutions for managing and alleviating the state space issue in Section 8.3.

Table 7.2: Stat space for timed RPL.

Topology	Initial RPL model	Timed RPL model
1	34	222
2	2178	65770
3	3422	50369
4	2150	44784

7.5 Related Works

Extension modelling formalisms is a common practice in the formal methods field to strengthen processes and, hence, modelling a wider range of systems. Several formalisms have been extended, with the intention of modelling real-time systems, by adding clocks into them. The next section provides some examples of such works, followed by a discussion of various works that model time aspects in bigraphs.

7.5.1 Real-Time Formalisms

Many modelling formalisms have been extended to real-time systems by introducing clocks. Timed CCS [160] extends Milner's CCS by introducing the notion of time to create concurrency models for real-time systems supporting both deterministic and non-deterministic behaviours. Timed CCS introduces another variable to record time delays, *e.g.*, before a message arrives. Similar to our work, timed CCS typically considers positive real numbers, including zero, as the time domain for convenience, but the model can deal with another numerical domain, *e.g.*, natural numbers. Timed Petri nets [166] extend Petri nets to allow time-triggered transitions. It uses tokens to allow transitions to start and then they are placed into the output when the firing process terminates. These authors used random variables with continuous or discrete probability distribution functions for the firing time. Timed π-Calculus [122] extends π-Calculus with continuous time to describe and reason about concurrent cyber-physical systems and real-time systems. An executable operational semantics of π-Calculus is developed in logic programming to model concurrency.

Graph transformation systems (GTS) have extended into probabilistic timed graph transformation systems (PTGTSs), which enable the modelling and analysing of structure dynamics and timed and probabilistic behaviour of embedded systems [93]. The clock is a typed node that is contained in a graph to identify the nodes that are used for time measurement only. In this work, PTGTSs are formally mapped into probabilistic timed automata (PTA), where the probabilistic timed structure (PTS) of the mapped PTGTSs corresponds to the PTS of the resulting PTA. Hence, they both satisfy the same set of PTCTL properties. The obtained PTA can be checked using PRISM. However, the mapping process does not consider the three aspects of the PTA. First, since the PTA does not consider valuations in the labelling function, constraints are ignored in the mapping process, so the constraint should be true for any such atomic proposition. It also considers the PTGTS that does not show timelocks during its execution, instead of finding and removing them when mapping PTGTS into the PTA. Finally, the GTS state space that is constructed for PTGTS is dependent on isomorphism as preserved clock nodes are respected, which may affect the state space finiteness of the resulting PTA. In our work, we bound the clock valuations, which ensures finite transition systems. Additionally, in the case that a system frequently resets its local clocks, the employment of the global clock guarantees the finiteness of the transition system. To prevent timelocks and, as a result, obtain a proper transition between reachable states, we assign the maximum states with invariant valuation to the rules that have a higher priority.

7.5.2 Modelling Time Aspects in Bigraphs

Bigraphs have been applied to model various timing aspects, for example, they have been used to model cloud systems that involve time constraints [161]. The cloud model includes aspects such as power consumption, mobile location and latency. The latter work performs an analysis by associating the reaction rule with the time of tasks at the evaluation step for cloud systems. BigrTimo [154] combines rTiMO process algebra and bigraphs to model the location and connectivity of the components of structure-aware mobile systems. BigrTimo uses real-time constraints to control actions by showing the waiting time for communication. These works do not model time constraints, such as those that we model in the examples above.

However, another work explicitly encodes clocks as entities within bigraphs to model and reason about cloud applications that share some similarities with our approach [82]. It adds a set of clocks and a set of clock constraints that are associated with nodes. It then utilises two different types of rules: (1) a set of reaction rules to advance all the clocks, so that all the clocks are advanced at the same speed, and (2) instantaneous rules that are executed only when there is a match and the time constraints of one or many nodes are satisfied. These rules can also update the clock constraints and reset the clocks. When a new time constraint is satisfied, another instantaneous rule may subsequently apply. The work confines the use of clocks to entities where a clock is nested in an entity. This results in encoding multiple reaction rules to advance clocks, which may cause unnecessary state-space explosion. It also does not provide a semantic definition that reflects the wall-time. Unlike our approach, it does not consider the non-deterministic behaviour that real-time systems often have. The work uses real-time Maude language [32] and its TCTL model checker is utilised to implement and analyse the approach. In contrast to their work, we encode timed aspects as action bigraphs that result in an MDP transition system that explicitly models the non-deterministic behaviour of timed systems, which can be formally checked by various model checking tools. The likelihood of a state space explosion is reduced here by employing a strategy that models all the clocks as a separate region, allowing us to advance all the clocks simultaneously. We can imitate the wall-time by utilising the global clock entity.

7.6 Summary

This chapter showed that bigraphs are extendable formalisms that are capable of modelling real-time systems by providing a modelling technique that introduces clocks into them. Here, the underlying transition system is an MDP that models the digital approximation (Section 7.1).

We illustrated the applicability of our approach through examples: PTA and a cloud system (Section 7.2). Moreover, Section 7.3 thoroughly described how to extend the initial RPL model of Chapter 5 with the timed aspects, *i.e.*, modelling of the DIS message. We discussed the limitations and thoughts of extending bigraphs with clocks in Section 7.4 while we provide, in Section 7.5, a brief overview of how other formalisms have been extended to model system timing behaviour. In the latter, we also included a number of other works that utilised bigraphs when modelling the timing aspects of systems.

Chapter 8

Discussion and Conclusion

This chapter concludes the thesis by highlighting the main research findings and contributions and provides recommendations for further research. Section 8.1 presents the thesis contributions while Section 8.2 discusses the limitations that lead to several directions for recommendations and possible extensions of this research (Section 8.3).

8.1 Contributions

This research has considered using bigraphs to formally model and analyse WSN routing protocols. Since RPL is one of the main routing protocols for WSNs, it has been used to showcase the proposed approach. In addition, as mentioned RPL shares some common features with other protocols. Hence, they can benefit from applying the same approach. Furthermore, by introducing clock notion to bigraphs theory, bigraphs are able to consider the timing aspects of routing protocols among other real-time systems.

This thesis contributes to formal modelling and WSN domains as follows:

- Novel RPL bigraphical models have been implemented using BigraphER toolkit. First, the initial model consists of 16 reaction rules (including two parameterised rules) encoding the four main steps of the RPL DAG constructions. The initial model considers the following. The model considers downwards connection where newly joined nodes multicast DIOs to their neighbours. It does not model DIS messages, which let a node request information upwards, as these require clocks, *e.g.*, try another DIS after n seconds, which was not supported by bigraphs. The other assumption is that nodes receive one DIO at a time, *i.e.*, that the processing time is sufficiently short relative to the message transmission time, and it is assumed that no packets drop in the network. Note that, because bigraphs support full model checking, all the possible execution traces, *i.e.*, message orders, are considered even if two or more messages appear at the same time. In addition, no node failures occur during DAG construction. These assumptions keep the initial model small to allow for efficient verification and to test the bigraph's ability to extend. However, some assumptions are removed in the following models, but they all have no effect on the construction of valid DAGs.

The model is formally analysed and checked using the PRISM model checker. A Python script was encoded to generate numerous network topologies that are used as an initial state to ensure the RPL model finds all the valid routes among different topologies. Three key properties were verified for each topology against the RPL model using the script that extracts the interesting graph features by leveraging the NetworkX library. Using bigraph patterns, interested states were labelled and then used to specify properties. The properties verify that the nodes eventually join the DAG; they join with optimal ranks and the constructed routes are cycle-free.

The assumptions are then removed by adding additional reaction rules to the second model. First, the DAO-ACK message was not considered since the assumption was that DAO-ACKs will always be successful. However, the model is then extended by adding $four$ reaction rules that consider the different possibilities of the message, which are given as follows. The DAO-ACK is sent (a) from the root to the node that is willing to join, (b) from the root to its child, which is a parent of the node willing to join, (c) from a parent to another one in the preferred parent list of the node willing to join; (d) from the preferred parent of the node willing to join directly. Another extension is concerned with the link and message receiving failures, which is modelled by adding an extra rule for each. The feasibility of the model extension is also examined by adding *three* popular WSN attacks; namely, a sinkhole attack, a hello-flood attack and a blackhole attack. These attacks are modelled by adding one reaction rule for each attack.

The belief is that other routing protocols, *e.g.*, RIP [121] or LEACH [52], can be modelled using a similar approach by utilising common features between protocols. In addition, the modelling of multiple protocols could beneficially determine a set of common approaches for the modelling protocols in general.

- The initial model was verified against RPL-Lite, a popular RPL implementation in the Contiki-Ng operating system using the Cooja simulator. The Python script was able to generate random network topologies in both formats: one in BigraphER syntax and one acceptable for Cooja. An experiment covering 100 topologies was performed. For each topology, BigraphER was run once and Cooja was repeatedly run for 500 times. The results reveal that, although the simulation may offer faster insights in some cases, BigraphER was more exhaustive. In addition, BigraphER identifies more possibly valid DAGs than Cooja, even with an extensive number of simulations. In one case, BigraphER found more than 14 valid routes.

  The modelling assumptions listed above are appropriate when comparing against RPL-Lite are as follows. The timing aspects (*i.e.*, clocks) of RPL-Lite are not considered since they do not play any role in the construction of the initial DAG, which is the focus of the initial model. In addition, nodes in Cooja can send DIS messages periodically, which is not the case for the bigraphical model. DIS messages were mainly used for DAG repair, which was not the focus of the initial model. That is, the presence of DIS does not enable different DAGs to be constructed compared with DIO only. Furthermore, 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 the orderings, this will result in finding all the possible parents regardless of link quality or the metric employed. One way to view this is that we consider all the possible link quality alternatives. Note that, as bigraph models are easily extendable, we can consider different objective functions. For instance, we can consider link quality by assuming different metrics for different quality measurements, *e.g.*, $q \in \{0,1,2,...\}$, and then employ a similar technique to the one used for ranks, comparing them to nominate the best link.

- Since modelling RPL timing aspects would require extensions to the bigraph theory, *i.e.*, introducing timers/real-time bigraphs, this thesis expresses clocks within action bigraphs to enable bigraphs to model real-time systems. We introduce a clock, as a new bigraph entity, for each timed entity in the system and we place these in a separate region so all the clocks can be manipulated without polluting the actual system model. This enables one reaction rule to advance all the clocks at once, which reduces the state space overhead of this approach. A set of reaction rules to model clock constraints in real-time systems through a digital clock approximation was defined.

  This approach allows for the explicit modelling of the non-determinism feature of real-time systems, *i.e.*, there is a choice between taking action or allowing time to pass at each state when the constraints are not yet satisfied. Chapter 7 illustrates how to use the proposed approach in practice by providing a model for PTA and requests allocation in a cloud system. Importantly, some timed aspects of the routing protocol for low-power and lossy networks (RPL) were modelled. An implementation of this approach is given in the BigraphER toolkit, to show that this is not just a theoretical contribution but a practical one.

8.2 Discussion

Simulation is the most common validation approach used by protocol designers. However, on comparing the formal model (an RPL bigraphs model) and the simulation (an RPL-Lite implementation), our results show an agreement with, *e.g.*, ref. [46, 47, 73], which is that the formal modelling coverage is greater than the simulation, as illustrated in Chapter 6. Although the bigraph model requires much more time than simulation, we believe that it is worth the wait since it examines the protocol against all of the possible outcomes, including rare events that are difficult to uncover with simulation. However, simulation provides quicker insight into how a protocol behaves under various network conditions, especially for large topology instances where state space explosion issues are present for the formal verification approach. In short, employing both approaches might be best practice for protocol design and validation, as described below. Bigraph models are used at a very early stage in the protocol design since they do not require the readiness of specifications, and they are easy to extend while the design is constructed because they do not require a full implementation effort. More importantly, while modelling is a quick way to try new ideas, it also provides robust answers, e.g., for all the possible DAGs. Simulation, on the other hand, allows experimentation at a much larger scale and a quick examination of different design choices and directions.

Modelling routing protocols first requires modelling of the network topologies. We noted that it is possible to provide a theoretical bigraphical model for WSN topologies by “drawing” the topologies with the required user-defined entities. However, to show the model in practice, BigraphER limits dealing with larger topologies. In addition to the effect of the number of nodes, the density of the topology increases the time required to construct the transition systems. Moreover, due to the dynamic configuration of the routing protocols, the modelling of routing protocols themselves can result in a state space explosion. However, we showed the accuracy of the model using a static analysis, as illustrated in Chapter 6. That is, although a static analysis is performed manually, it mitigates the limitation of automatic model checking by verifying the model through validation of the reaction rules regardless of the number of nodes and the topology.

Utilising bigraphs, in particular, in the modelling and verification of routing protocols, shows various benefits. Despite the fact that protocol design benefits from formal methods in general, it may still not be completely satisfactory for designers who do not have a strong mathematical background. Bigraphs, as a diagrammatic formalism, can be seen as easier to understand than other heavily mathematical formalisms and, thus, leveraging such diagrammatic modelling framework opens up formal verification to a wider audience. As Section 5.3.2 showed, the RPL standard is a long and detailed document, has multiple supported documents and imposes some ambiguities that natural languages normally have. Therefore, modelling protocols from such standards was not an easy task. We believe that using bigraphs in the early design stage facilitates capturing such complex and ambiguous descriptions. Furthermore, bigraph models are easy to extend and do not require full implementation efforts. The verification process can be gradually performed while specifying the protocol document, which enables early discovery of design flaws and bugs that lead to a more cost-effective resolution.

A notable feature of bigraphs is that they are extendable as a formalism. Introducing clocks, theoretically, to the bigraphs was applicable. We take advantage of action BRSs, which is an extended variant of bigraphs, to propose a technique that allows for the modelling of timing aspects in several systems. We show that our approach supports multiple clocks and the transition system, obtained via rewriting, faithfully encodes the digital-clock approximation of the behaviour of a real-time system. We mitigate a state space explosion by adopting the following two strategies. First, we bound the clock valuations; thus, we obtain finite transition systems. Second, we allow the base time unit to be specified in each model, effectively allowing clocks to advance by multiple ticks in one step. However, the BigraphER toolkit struggles with state space explosion when modelling timed RPL. As discussed in Section 7.4, this might not be the case for all real-time systems, as we have shown the practical application of the proposed technique in a PTA and a cloud system.

8.3 Future Work

This section presents several directions for future work derived from the discussion and limitations presented in Section 8.2, and possible extensions of this research are also presented. In addition, this section highlights possible publications that could be produced based on the research in this thesis.

8.3.1 Directions

The following provides various areas that are worthy of investigation, driven by Section 8.2:

- A translation tool between simulators, as a preferable technique for engineers, and a formal method since it is proven to have more coverage.

- Develop syntactic support for clock constraints in the BigraphER language to generate real-time ABRS models.

- Develop automated inductive reasoning tool that verifies the model through a validation of the reaction rules, considering priorities and conditions.

- Provide BigraphER with a model checking support that considers how BigraphER generates system transitions. In addition, support the tool with appropriate state space mitigation techniques.

- Support utilising formal modelling formalisms, such as bigraphs, that are easily extended in an early stage of the protocol specification, while designing protocols (not afterwards), *i.e.*, providing organisations such as IETF with research proposals that aim at improving protocol design. That will result in earlier maintaining ambiguity and discovering inconsistency.

- Conduct a case study to explore how protocol designers use bigraphs as a verification technique for protocols.

8.3.2 Planned Publications

The research in this thesis has already appeared in *two* peer-reviewed publications. However, there exists a solid possibility for more publications, as explained below.

- One of the published papers presents preliminary results on the extension of bigraphs with the use of clocks. The paper briefly shows that the RPL time aspects can be modelled with the proposed technique. In addition, the other paper presents the initial model for the RPL protocol. Hence, the plan is to include the entire RPL timed model and extensions, as presented in Chapters 5 and 7.

- The RPL model is extendable and its ability to include security attacks is illustrated in Chapter 5. Therefore, the plan is to formally analyse and reason about security attacks that threaten RPL to investigate how bigraphs detect and prevent WSN vulnerabilities.

Appendix A

Initial RPL Bigraphs Models

```
ctrl Physical = 0;
ctrl PNode = 1;
ctrl PLink = 1;
ctrl Routing = 0;
ctrl Node = 1;
atomic fun ctrl ID(i) = 0;
atomic ctrl PrefPrnt= 1;
atomic ctrl Idle = 0;
atomic ctrl In_use = 0;
atomic ctrl ClearMulticast = 0;
ctrl DIO = 1;
atomic ctrl DAO = 2;
ctrl Send = 0;
ctrl Receive = 0;
atomic ctrl Joined=0;
atomic ctrl GenDIO = 0;
ctrl Rank = 0;
atomic fun ctrl Val(n) = 0;
ctrl IncRank = 0;
ctrl LT = 0;
ctrl L = 0;
ctrl R = 0;
atomic ctrl False = 0;
atomic ctrl True = 0;

fun react incRank(m) =
   IncRank.Rank.Val(m)
 -[1]->
   Rank.Val(m+1);

fun react ranksComparison(n,m) =
  LT.(L.Rank.Val(n) | R.Rank.Val(m))
  -[1]->
  LT.(L.Rank.Val(n-1) | R.Rank.Val(m-1));

react lt_l_0 =
    LT.(L.Rank.Val(0) | id )
  -[1]->
    LT.True @[];
react lt_r_0 =
   LT.(R.Rank.Val(0) | id )
 -[1]->
   LT.False @[];

react generateDIO =
  Node{n}.(Rank.id | GenDIO | id)
  -[1]->
  Node{n}.(Rank.id | Send.DIO{n}.Rank.id | id) @[0,0,1];

react sendDIO =
   (Node{n1}.(Send.DIO{n}.id | id) | Node{n2}.id)
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.id | id))
-[1]->
   (Node{n1}.(Send.DIO{n}.id | id) | Node{n2}.(Receive.DIO{n}.id | id))
|| /x (PNode{n1}.(PLink{x}.In_use | id) | PNode{n2}.(PLink{x}.id | id)) @ [0,1,0,2,3,4,5];

react sendDIO_Done =
  Node{n}.(Send.DIO{n}.id | id) || PNode{n}.id
  -[1]->
  Node{n}.id || PNode{n}.(ClearMulticast | id) @[1,2] if !Idle in param;

react clearMulticast =
  PNode{n}.(ClearMulticast | PLink{x}.In_use | id)
  -[1]->
  PNode{n}.(ClearMulticast | PLink{x}.Idle | id);

react clearMulticastEnd =
  PNode{n}.(ClearMulticast | id)
  -[1]->
  PNode{n}.id  if !In_use in param;

react handleDIONotJoined =
  Node{n}.(Receive.DIO{p}.Rank.id | id)
  -[1]->
  Node{n}.(IncRank.Rank.id | PrefPrnt{p} | Send.DAO{n,p} | id) if !Rank in param;

react handleDIOJoined =
  Node{n}.(Rank.id | Receive.DIO{p}.Rank.id | id)
  -[1]->
  Node{n}.(Rank.id | Receive.DIO{p}.Rank.id | LT.(L.Rank.id | R.IncRank.Rank.id) | id) @
      [0,1,0,1,2] if !LT in param;

react handleDIOLT =
  Node{n}.(Receive.DIO{p}.Rank.id | LT.True | id)
  -[1]->
  Node{n}.id | {p} @[1];

react handleDIOGT =
  Node{n}.(Receive.DIO{p}.Rank.id | Rank.id | LT.False | PrefPrnt{x} | id)
  -[1]->
  Node{n}.(IncRank.Rank.id | PrefPrnt{p} | Send.DAO{n,p} | id) | {x} @[0,2];

react sendDAO =
    (Node{n1}.(Send.DAO{n1,n2} | id) | Node{n2}.id)
-[1]->
    (Node{n1}.(GenDIO | Joined | id) | Node{n2}.(Receive.DAO{n1,n2} | id));

react sendDAOUpParent =
  Node{n1}.(Receive.DAO{n1,f} | PrefPrnt{n2} |id) | Node{n2}.id
```

```
-[1]->
  Node{n1}.(  PrefPrnt{n2} | id )  | Node{n2}.(Receive.DAO{n1,n2} | {f} | id);

react clearDAO =
   Node{n1}.(Receive.DAO{n1,n2} | id)
-[1]->
   Node{n1}.( id | {n1} | {n2}) ;

#########################
big autoGenerated =
 /n0/l0/l1/l2/l3/n1/n2/n3 (
 Physical.( PNode{n0}.(PLink{l0}.Idle | PLink{l1}.Idle) | PNode{n1}.(PLink{l0}.Idle | PLink{
     l3}.Idle) | PNode{n2}.(PLink{l1}.Idle | PLink{l2}.Idle) | PNode{n3}.(PLink{l2}.Idle  |
     PLink{l3}.Idle ) ) ||
 Routing.( Node{n0}.(ID(1) | Rank.Val(0) | GenDIO ) | Node{n1}.(ID(2)  ) | Node{n2}.(ID(3)
        ) | Node{n3}.(ID(4)  ) )
);
 ########################

# Predicates
fun big rank(i,n) = Node{n1}.(ID(i) | Rank.Val(n) | id );

fun big parentOf(i',i) = Node{n1}.( ID(i') | PrefPrnt{n2}  | Joined | id ) | Node{n2}.( ID(i)
     | id);

fun big joined(i) = Node{n1}.( ID(i) | PrefPrnt{n2}  | Joined | id ) ;

fun big  multijoin(i) = Node{n}.(Rank.id | Rank.id | ID(i) | id);


begin pbrs

  int n={0,1,2,3,4,5,6,7,8,9,10};
  int n'={1,2,3,4,5,6,7,8,9,10};
  int m={0,1,2,3,4,5,6,7,8,9,10};
  int m'={1,2,3,4,5,6,7,8,9,10};
  int i = {1, 2, 3, 4, 5, 6,7,8,9};
  int i' = {1, 2, 3, 4, 5, 6,7,8,9};


init autoGenerated;

  rules = [
  (handleDIONotJoined, handleDIOJoined, handleDIOLT, handleDIOGT),
  {sendDIO},
  (incRank(n),lt_l_0, lt_r_0, ranksComparison(n',m')),
  (sendDIO_Done, clearMulticast, clearMulticastEnd),
  (sendDAOUpParent),
  {sendDAO},
  (clearDAO),
  (generateDIO)
  ];

preds = {parentOf(i',i), rank(i,n), joined(i), multijoin(i)};

end
```

Appendix B

Timed RPL Bigraphs Models

```
ctrl Physical = 0;
ctrl PNode = 1;
ctrl PLink = 1;
ctrl Routing = 0;
ctrl Node = 1;
atomic fun ctrl ID(i) = 0;
atomic ctrl PrefPrnt= 1;
atomic ctrl Idle = 0;
atomic ctrl In_use = 0;
atomic ctrl Wait=0;
atomic ctrl Pass=0;
atomic ctrl Willing=0;
atomic ctrl Sent=0;
atomic ctrl Fail =0 ;
atomic ctrl ClearMulticast = 0;
ctrl DIO = 1;
atomic ctrl DAO = 2;
atomic ctrl DIS = 1;
atomic ctrl ACK = 2;
ctrl Send = 0;
ctrl Receive = 0;
atomic ctrl Joined=0;
atomic ctrl GenDIO= 0;
ctrl Rank = 0;
atomic fun ctrl Val(n) = 0;
ctrl IncRank = 0;
ctrl LT = 0 ;
ctrl L = 0 ;
ctrl R = 0 ;
atomic ctrl False = 0 ;
atomic ctrl True = 0 ;
ctrl LocalClock = 0;
fun ctrl LC(sensor1_Send_Time) = 1;
atomic fun ctrl GC(sensor1_Send_Time) = 0;
atomic ctrl Update = 0 ;
atomic ctrl Ticked = 0 ;
atomic ctrl To_tick = 0 ;
atomic ctrl Done = 0 ;
ctrl LocalClockDis =0;
ctrl LC_Dis = 1;
```

```
fun react incRank(m) =
   IncRank.Rank.Val(m)
-[1]->
   Rank.Val(m+1);

fun react ranksComparison(n,m) =
  LT.(L.Rank.Val(n) | R.Rank.Val(m))
-[1]->
  LT.(L.Rank.Val(n-1) | R.Rank.Val(m-1));

react lt_l_0 =
  LT.(L.Rank.Val(0) | id )
-[1]->
  LT.True @[];
react lt_r_0 =
  LT.(R.Rank.Val(0) | id )
-[1]->
  LT.False @[];

react clockReset = LocalClock.id
           -[1]->
                  LocalClockDis.Done @[] if !Willing in ctx;

react clockDisable =
 ( Node{n1}.id | Node{n2}.(Joined | id))
 || /x ( PNode{n1}.(PLink{x}.id | id) | PNode{n2}.(PLink{x}.id | id) )
 || LC(2){n2}.id
  -[1]->
 ( Node{n1}.id | Node{n2}.(Joined | id))
 || /x ( PNode{n1}.(PLink{x}.id | id) | PNode{n2}.(PLink{x}.id | id) )
 || LC_Dis{n2} if !Willing in param;

react updated_clock_advance =
   LocalClock.id
-[1]->
  LocalClock.(Update | id) if LC(0){c} in param, !LC(2){c} in param;

react updated_clock_advance1 =
 LocalClock.id
-[1]->
 LocalClock.(Update | id) if LC(1){c} in param, !LC(2){c} in param;

fun react tick(sensorClock) =
  LocalClock.(Update | LC(sensorClock){c}.To_tick | id)
  -[1]->
  LocalClock.(Update | LC(sensorClock + 1){c}.Ticked | id);

fun react tickDone(sending_Time) =
LocalClock.(LC(sending_Time){c}.Ticked | Update | id)
  -[1]->
 LocalClock.(LC(sending_Time){c}.To_tick | Update  | id) ;

react updated_clock_advance_Done =
  LocalClock.( Update | id )
-[1]->
  LocalClock.id if !Ticked in param;

fun react generateDIS(sending_Time) =
    Node{n}.(Willing | id) || LocalClock.( LC(sending_Time){n}.id | id )
-[1]->
   Node{n}.(DIS{n} | Willing | id ) || LocalClock.( LC(0){n}.id | id ) if !Sent in param ;

fun react sendDIS(clockVal) =
( Node{n1}.id | Node{n2}.(DIS{n2} | id))
|| /x ( PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.id | id) )
|| LC(clockVal){n1}.id
-[1]->
 ( Node{n1}.(Receive.DIS{n1} | id) | Node{n2}.(Sent | id))
|| /x ( PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.id | id) )
|| LC(1){n1}.id ;

react sendDIS_Fail =
 Node{n}.(Sent | Willing | id) || LC(1){n}.id
  -[1]->
 Node{n}.(Willing | Fail | id) || LC(0){n}.id ;

react receiveDIS =
  Node{n}.(Rank.id | Receive.DIS{n} | id)
|| LC(1){n}.id
  -[1]->
  Node{n}.(Rank.id | Send.DIO{n}.Rank.id | id) || LC(0){n}.id @[0,0,1,2];

react ignoreDIS =
  Node{n}.( Receive.DIS{n} | id)
-[1]->
  Node{n}.( id )  if !Joined in param;

react generateDIO =
  Node{n}.(Rank.id | GenDIO | id)
  -[1]->
  Node{n}.(Rank.id |  Send.DIO{n}.Rank.id | id)   @[0,0,1] ;

react multicastDIO =
   (Node{n1}.(Send.DIO{n}.id | id) | Node{n2}.(Joined | id))
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.id | id))
  -[1]->
   (Node{n1}.( Send.DIO{n}.id | id) | Node{n2}.( Joined | Receive.DIO{n}.id  | id))
|| /x (PNode{n1}.(PLink{x}.In_use | id) | PNode{n2}.(PLink{x}.id | id)) @ [0,1,0,2,3,4,5];

react sendDIO_To_FailNodes =
   (Node{n1}.(Send.DIO{n}.id | id) | Node{n2}.(Fail | id))
|| /x (PNode{n1}.(PLink{x}.id | id) | PNode{n2}.(PLink{x}.id | id))
-[1]->
   (Node{n1}.( Send.DIO{n}.id | id) | Node{n2}.( Receive.DIO{n}.id  | id))
|| /x (PNode{n1}.(PLink{x}.id | id) | PNode{n2}.(PLink{x}.id | id)) @ [0,1,0,2,3,4,5,6];

react sendDIO =
   (Node{n1}.(Send.DIO{n}.id | id) | Node{n2}.(Sent |  id))
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.id | id))
-[1]->
   (Node{n1}.( Send.DIO{n}.id | id) | Node{n2}.( Receive.DIO{n}.id  | id))
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.id | id)) @ [0,1,0,2,3,4,5] ;

react sendDIO_Done =
  Node{n}.(Send.DIO{n}.id  | id) || PNode{n}.id
  -[1]->
  Node{n}.id || PNode{n}.(ClearMulticast | id) @[1,2] if !Idle in param; # Condition, Sent to
       everyone

react clearMulticast =
  PNode{n}.(ClearMulticast | PLink{x}.In_use | id)
  -[1]->
  PNode{n}.(ClearMulticast | PLink{x}.Idle | id);

react clearMulticastEnd =
  PNode{n}.(ClearMulticast | id)
  -[1]->
  PNode{n}.id if !In_use in param; # Finished reverting everything to idle

react handleDIONotJoined =
  Node{n}.(Receive.DIO{p}.Rank.id |  id)
  -[1]->
  Node{n}.(IncRank.Rank.id | PrefPrnt{p} | Send.DAO{n,p} | id)
  if !Rank in param;

react handleDIOJoined =
  Node{n}.(Rank.id | Receive.DIO{p}.Rank.id |  id)
  -[1]->
  Node{n}.(Rank.id | Receive.DIO{p}.Rank.id | LT.(L.Rank.id | R.IncRank.Rank.id) | id) @
      [0,1,0,1,2] if !LT in param;

react handleDIOLT =
  Node{n}.(Receive.DIO{p}.Rank.id | LT.True | id)
  -[1]->
  Node{n}.id | {p} @[1];

react handleDIOGT =
  Node{n}.(Receive.DIO{p}.Rank.id | Rank.id | LT.False | PrefPrnt{x} | id)
  -[1]->
  Node{n}.(IncRank.Rank.id | PrefPrnt{p} | Send.DAO{n,p} | id) | {x}   @[0,2];

react sendDAO =
    (Node{n1}.(Send.DAO{n1,n2} | id) | Node{n2}.id)
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Idle | id))
 -[1]->
    (Node{n1}.( id) | Node{n2}.(Receive.DAO{n1,n2} | id))
|| /x (PNode{n1}.(PLink{x}.Wait | id) | PNode{n2}.(PLink{x}.Idle | id));

react sendDAOUpParent =
  Node{n1}.(Receive.DAO{n1,f} | PrefPrnt{n2} | id) | Node{n2}.id
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.id | id))
     -[1]->
   Node{n1}.(  PrefPrnt{n2} | id )  | Node{n2}.(Receive.DAO{n1,n2} | {f} | id)
|| /x (PNode{n1}.(PLink{x}.Pass | id) | PNode{n2}.(PLink{x}.id | id));

react sendACKRootParent =
    Node{n1}.(Receive.DAO{n1,n2} | id) | Node{n2}.(PrefPrnt{n1} | id )
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Pass | id))
 -[1]->
    Node{n1}.( Send.ACK{n1,n2} | id) | Node{n2}.(PrefPrnt{n1} | Receive.ACK{n1,n2} |  id )
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Idle | id));

react sendACKRootNode =
  Node{n1}.(Receive.DAO{n1,n2}  | id) | Node{n2}.(PrefPrnt{n1} | id )
```

```
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Wait | id))
 -[1]->
  Node{n1}.( Send.ACK{n1,n2} | id) | Node{n2}.(PrefPrnt{n1} | Receive.ACK{n1,n2} |  id )
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Idle | id));

react sendACKParentParent =
     Node{n1}.(Receive.ACK{n1,p} | id ) | Node{n2}.( id)
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Pass | id))
          -[1]->
    Node{n1}.(id | {p}) | Node{n2}.( Receive.ACK{n1,n2}  | id)
 || /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Idle | id));

react sendACKParentNode =
   Node{n1}.(Receive.ACK{n1,p} | id ) | Node{n2}.( id)
 || /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Wait | id))
-[1]->
   Node{n1}.(id | {p}) | Node{n2}.( Receive.ACK{n1,n2}  | id)
|| /x (PNode{n1}.(PLink{x}.Idle | id) | PNode{n2}.(PLink{x}.Idle | id));

react toJoin =
  Node{n}.(Receive.ACK{n,p} | Willing | id )
-[1]->
  Node{n}.( Joined | GenDIO | id) | {p} ;

react clearDAO =
  Node{n1}.(Receive.DAO{n1,n2} | id)
-[1]->
  Node{n1}.( id | {n1} | {n2}) ;

react clear_Sent_Ack = Node{n1}.(Send.ACK{n1,n2} | id)
-[1]->
                 Node{n1}.( id | {n1} | {n2}) ;

react clear_Recieved_ACK =
 Node{n1}.(Receive.ACK{n1,n2} | id)
-[1]->
 Node{n1}.( id | {n1} | {n2}) ;

react clearDIS = Node{n}.(Receive.DIS{n} | id ) -[1]-> Node{n}.id if !Willing in ctx;

big autoGenerated =
 /n0/l0/l1/l2/l3/n1/n2/n3 (
 Physical.( PNode{n0}.(PLink{l0}.Idle | PLink{l1}.Idle) | PNode{n1}.(PLink{l0}.Idle | PLink{
     l3}.Idle) | PNode{n2}.(PLink{l1}.Idle | PLink{l2}.Idle) | PNode{n3}.(PLink{l2}.Idle  |
     PLink{l3}.Idle ) ) ||
 Routing.( Node{n0}.(ID(1) | Rank.Val(0) | GenDIO | Joined) | Node{n1}.(ID(2) | Willing ) |
     Node{n2}.(ID(3) | Willing   ) | Node{n3}.(ID(4) | Willing ) ) ||
 LocalClock.(  LC(0){n0}.To_tick |  LC(0){n1}.To_tick |   LC(0){n2}.To_tick |  LC(0){n3}.
     To_tick )
);

fun big sensor_Clock(i, n) = LocalClock.( LC(n){n0} | id ) || Node{n0}.(ID(i) | id );

fun big parentOf(i',i) = Node{n1}.( ID(i') | PrefPrnt{n2}  | Joined | id ) | Node{n2}.( ID(i)
     | id);

fun big rankof(i,n) = Node{n1}.( ID(i) | Rank.Val(n) | id );
```

```
begin abrs
  int n={0,1,2,3,4,5,6,7,8,9,10};
  int n'={1,2,3,4,5,6,7,8,9,10};
  int m={0,1,2,3,4,5,6,7,8,9,10};
  int m'={1,2,3,4,5,6,7,8,9,10};
  int i = {1, 2, 3, 4, 5, 6,7,8,9};
  int i' = {1, 2, 3, 4, 5, 6,7,8,9};

  int sensorClock={0,1};
  int sending_Time={1,2};
  int clockVal ={0,1,2};

init autoGenerated;

  rules = [
   (toJoin),
   (sendDIO_Done, clearMulticast, clearMulticastEnd),
   (incRank(n),lt_l_0, lt_r_0, ranksComparison(n,m)),
   (handleDIONotJoined, handleDIOJoined, handleDIOLT, handleDIOGT),
   (sendACKParentParent, sendACKRootParent),
   (multicastDIO, sendDIO_To_FailNodes),
   {sendDAO},
   (sendDAOUpParent),
   {sendACKParentNode, sendACKRootNode},
   (clearDAO, clear_Sent_Ack, clear_Recieved_ACK),
   (sendDIS(clockVal)),
   (ignoreDIS, receiveDIS),
   (generateDIO),
   (tick(sensorClock)), (tickDone(sending_Time),updated_clock_advance_Done),
   (clearDIS),
   (clockReset, clockDisable),
   {sendDIO, sendDIS_Fail},
   {generateDIS(sending_Time), updated_clock_advance,updated_clock_advance1}
];

actions=[
  tick={updated_clock_advance, updated_clock_advance1, tick, tickDone,
      updated_clock_advance_Done},
  reset={clockReset, clockDisable},
  sendDIS = {generateDIS, sendDIS, sendDIS_Fail},
  sendingDIO={sendDIO},
  handleDIS= {receiveDIS, ignoreDIS},
  handleDIO={handleDIONotJoined, handleDIOJoined, handleDIOLT, handleDIOGT},
  calculateRank ={incRank,lt_l_0, lt_r_0, ranksComparison},
  sendDAO={sendDAO, sendDAOUpParent},
  sendACK = {sendACKParentParent, sendACKParentNode, sendACKRootParent, sendACKRootNode},
  join = {toJoin},
  advertise={generateDIO, multicastDIO, sendDIO_To_FailNodes},
  clearMSGs={clearDIS, clearDAO, clear_Sent_Ack, sendDIO_Done, clearMulticast,
      clearMulticastEnd, clear_Recieved_ACK}
];

   preds = {parentOf(i',i), sensor_Clock(i, clockVal), rankof(i,n)};
end
```

Bibliography

[1] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010. ISBN: 978-0-521-89556-9. DOI: 10.1017/CBO9781139195881 URL: https://doi.org/10.1017/CBO9781139195881.

[2] 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.

[3] Vaibhav Ajayy and Virender Ranga. “Performance analysis of RPL protocol in different nodes positioning using Contiki Cooja”. In: *International Journal of Information Technology* (2024), pp. 1–7.

[4] Taief Alaa Alamiedy et al. “A systematic literature review on attacks defense mechanisms in RPL-based 6LoWPAN of Internet of Things”. In: *Internet Things* 22 (2023), p. 100741. DOI: 10.1016/J.IOT.2023.100741. URL: https://doi.org/10.1016/j.iot.2023.100741.

[5] 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.

[6] Rajeev Alur and David L. Dill. “A Theory of Timed Automata”. In: *Theor. Comput. Sci.* 126.2 (1994), pp. 183–235. DOI: 10.1016/0304-3975(94)90010-8.

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

[8] Blair Archibald, Muffy Calder, and Michele Sevegnani. “Practical Modelling with Bigraphs”. In: *CoRR* abs/2405.20745 (2024). DOI: 10.48550/ARXIV.2405.20745. arXiv: 2405.20745. URL: https://doi.org/10.48550/arXiv.2405.20745.

[9] 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.

[10] 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.

[11] Blair Archibald et al. "Modelling and verifying BDI agents with bigraphs". In: *Science of Computer Programming* 215 (2022), p. 102760. ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2021.102760. URL: https://www.sciencedirect.com/science/article/pii/S0167642321001532.

[12] 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.

[13] 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.

[14] Chris R. Baker et al. "Wireless Sensor Networks for Home Health Care". In: *21st International Conference on Advanced Information Networking and Applications (AINA 2007), Workshops Proceedings, Volume 2, May 21-23, 2007, Niagara Falls, Canada*. IEEE Computer Society, 2007, pp. 832–837. DOI: 10.1109/AINAW.2007.376. URL: https://doi.org/10.1109/AINAW.2007.376.

[15] Richard Bellman. "A Markovian decision process". In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[16] 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.

[17] Johan Bengtsson et al. "Partial Order Reductions for Timed Systems". In: *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*. Ed. by Davide Sangiorgi and Robert de Simone. Vol. 1466. Lecture Notes in Computer Science. Springer, 1998, pp. 485–500. DOI: 10.1007/BFB0055643. URL: https://doi.org/10.1007/BFb0055643.

[18] Cinzia Bernardeschi et al. "A framework for formal analysis and simulative evaluation of security attacks in wireless sensor networks". In: *J. Comput. Virol. Hacking Tech.* 17.3 (2021), pp. 249–263. DOI: 10.1007/S11416-021-00392-0. URL: https://doi.org/10.1007/s11416-021-00392-0.

[19] Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. “Formal verification of standards for distance vector routing protocols”. In: *J. ACM* 49.4 (2002), pp. 538–576. DOI: 10.1145/581771.581775. URL: https://doi.org/10.1145/581771.581775.

[20] Bharat Bhushan and G. Sahoo. “Routing Protocols in Wireless Sensor Networks”. In: *Computational Intelligence in Sensor Networks*. Ed. by Bijan Bihari Mishra et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 215–248. ISBN: 978-3-662-57277-1. DOI: 10.1007/978-3-662-57277-1_10. URL: https://doi.org/10.1007/978-3-662-57277-1_10.

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

[22] 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).

[23] Jerry R Burch et al. “Symbolic model checking: 1020 states and beyond”. In: *Information and computation* 98.2 (1992), pp. 142–170.

[24] 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.

[25] Louie Chan et al. “Hierarchical routing protocols for wireless sensor network: A compressive survey”. In: *Wireless Networks* 26 (2020), pp. 3291–3314.

[26] Zhe Chen et al. “A review of automated formal verification of ad hoc routing protocols for wireless sensor networks”. In: *Sensor Letters* 11.5 (2013), pp. 752–764.

[27] Maxim Chernyshev et al. “Internet of Things (IoT): Research, Simulators, and Testbeds”. In: *IEEE Internet Things J.* 5.3 (2018), pp. 1637–1647. DOI: 10.1109/JIOT.2017.2786639. URL: https://doi.org/10.1109/JIOT.2017.2786639.

[28] Alessio Chiapperini, Marino Miculan, and Marco Peressotti. “Computing Embeddings of Directed Bigraphs”. In: *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12150. Lecture Notes in Computer Science. Springer, 2020, pp. 38–56. DOI: 10.1007/978-3-030-51372-6_3. URL: https://doi.org/10.1007/978-3-030-51372-6%5C_3.

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

[30] 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.

[31] Edmund M. Clarke et al. “Model Checking and the State Explosion Problem”. In: *Tools for Practical Software Verification, LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*. Ed. by Bertrand Meyer and Martin Nordio. Vol. 7682. Lecture Notes in Computer Science. Springer, 2011, pp. 1–30. DOI: 10.1007/978-3-642-35746-6_1. URL: https://doi.org/10.1007/978-3-642-35746-6%5C_1.

[32] 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.

[33] Sylvain Conchon et al. “Cubicle: a parallel SMT-based model checker for parameterized systems: tool paper”. In: *Computer Aided Verification: 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings 24*. Springer. 2012, pp. 718–724.

[34] Troels C Damgaard et al. “An inductive characterization of matching in binding bigraphs”. In: *Formal Aspects of Computing* 25.2 (2013), pp. 257–288.

[35] Khalid A. Darabkh et al. “RPL routing protocol over IoT: A comprehensive survey, recent advances, insights, bibliometric analysis, recommendations, and future directions”. In: *J. Netw. Comput. Appl.* 207 (2022), p. 103476. DOI: 10.1016/J.JNCA.2022.103476. URL: https://doi.org/10.1016/j.jnca.2022.103476.

[36] 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 30*. Springer. 2017, pp. 592–600.

[37] Luming Dong et al. “Specifying and Verifying SDP Protocol Based Zero Trust Architecture Using TLA+”. In: *Proceedings of the 7th International Conference on Cyber Security and Information Engineering*, pp. 35–43.

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

[39] Hartmut Ehrig et al. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006. ISBN: 978-3-540-31187-4. DOI: `10.1007/3-540-31188-2`. URL: `https://doi.org/10.1007/3-540-31188-2`.

[40] Hossam M. A. Fahmy. *Concepts, applications, experimentation and analysis of wireless sensor networks*. English. Third. Cham: Springer Nature Switzerland, 2023. ISBN: 9783031207099.

[41] Ansgar Fehnker, Matthias Fruth, and AK McIver. "Graphical modelling for simulation and formal analysis of wireless network protocols". In: *Methods, Models and Tools for Fault Tolerance*. Springer, 2009, pp. 1–24.

[42] Emad A. Felemban et al. "Underwater Sensor Network Applications: A Comprehensive Survey". In: *Int. J. Distributed Sens. Networks* 11 (2015), 896832:1–896832:14. DOI: `10.1155/2015/896832`. URL: `https://doi.org/10.1155/2015/896832`.

[43] Amal Gassara, Ismael Bouassida, and Mohamed Jmaiel. "A tool for modeling sos architectures using bigraphs". In: *Proceedings of the Symposium on Applied Computing*. 2017, pp. 1787–1792.

[44] Baraq Ghaleb et al. "A Survey of Limitations and Enhancements of the IPv6 Routing Protocol for Low-Power and Lossy Networks: A Focus on Core Operations". In: *IEEE Commun. Surv. Tutorials* 21.2 (2019), pp. 1607–1635. DOI: `10.1109/COMST.2018.2874356`. URL: `https://doi.org/10.1109/COMST.2018.2874356`.

[45] 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.

[46] M Girish, G Gopakumar, and DS Divya. "Formal and Simulation Verification: Comparing and Contrasting the two Verification Approaches". In: *2021 2nd International Conference on Advances in Computing, Communication, Embedded and Secure Systems (ACCESS)*. IEEE. 2021, pp. 41–44.

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

[48] Omprakash Gnawali and Philip Levis. *Rfc 6719: The minimum rank with hysteresis objective function*. 2012.

[49] Davide Grohmann and Marino Miculan. "Directed bigraphs". In: *Electronic Notes in Theoretical Computer Science* 173 (2007), pp. 121–137.

[50] Hans Hansson and Bengt Jonsson. "A Logic for Reasoning about Time and Reliability". In: *Formal Aspects Comput.* 6.5 (1994), pp. 512–535. DOI: `10.1007/BF01211866`.

[51] Arnd Hartmanns and Holger Hermanns. “Explicit Model Checking of Very Large MDP Using Partitioning and Secondary Storage”. In: *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*. Ed. by Bernd Finkbeiner, Geguang Pu, and Lijun Zhang. Vol. 9364. Lecture Notes in Computer Science. Springer, 2015, pp. 131–147. DOI: `10.1007/978-3-319-24953-7\_10`. URL: `https://doi.org/10.1007/978-3-319-24953-7%5C_10`.

[52] Wendi B. Heinzelman, Anantha P. Chandrakasan, and Hari Balakrishnan. “An application-specific protocol architecture for wireless microsensor networks”. In: *IEEE Trans. Wirel. Commun.* 1.4 (2002), pp. 660–670. DOI: `10.1109/TWC.2002.804190`. URL: `https://doi.org/10.1109/TWC.2002.804190`.

[53] Martijn Hendriks et al. “Adding Symmetry Reduction to Uppaal”. In: *Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6-7, 2003. Revised Papers*. Ed. by Kim Guldstrand Larsen and Peter Niebert. Vol. 2791. Lecture Notes in Computer Science. Springer, 2003, pp. 46–59. DOI: `10.1007/978-3-540-40903-8\_5`. URL: `https://doi.org/10.1007/978-3-540-40903-8%5C_5`.

[54] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. “What Good Are Digital Clocks?” In: *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*. Ed. by Werner Kuich. Vol. 623. Lecture Notes in Computer Science. Springer, pp. 545–558.

[55] Katharina Hofer-Schmitz and Branka Stojanović. “Towards formal verification of IoT protocols: A Review”. In: *Computer Networks* 174 (2020), p. 107233.

[56] Espen Højsgaard and Arne John Glenstrup. *The BPL Tool-a Tool for Experimenting with Bigraphical Reactive Systems*. IT University of Copenhagen, 2011.

[57] 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`.

[58] Ronald A Howard. “Dynamic programming and markov processes.” In: (1960).

[59] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. “The coq proof assistant a tutorial”. In: *Rapport Technique* 178 (1997).

[60] Michael Huth and Mark Dermot Ryan. *Logic in computer science - modelling and reasoning about systems (2. ed.)* Cambridge University Press, 2004.

[61] Graham Hutton. “Book Review: Introduction to HOL: A Theorem Proving Environment for Higher Order Logic by Mike Gordon and Tom Melham (eds.), Cambridge University Press, 1993, ISBN 0-521-44189-7”. In: *J. Funct. Program.* 4.4 (1994), pp. 557–559. DOI: 10.1017/S0956796800001180. URL: https://doi.org/10.1017/S0956796800001180.

[62] IDC. *Future of Industry Ecosystems: Shared Data and Insights*. Accessed: 2024-12-01. Jan. 2021. URL: https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/.

[63] 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.

[64] Ashish R Jadhao and Sharwari S Solapure. “Analysis of routing protocol for Low Power and Lossy Networks (RPL) using Cooja simulator”. In: *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE. 2017, pp. 2364–2368.

[65] Philo Juang et al. “Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet”. In: *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), San Jose, California, USA, October 5-9, 2002*. Ed. by Kourosh Gharachorloo and David A. Wood. ACM Press, 2002, pp. 96–107. DOI: 10.1145/605397.605408. URL: https://doi.org/10.1145/605397.605408.

[66] Patrick Olivier Kamgueu, Emmanuel Nataf, and Thomas Ndié Djotio. “Survey on RPL enhancements: A focus on topology, security and mobility”. In: *Comput. Commun.* 120 (2018), pp. 10–21. DOI: 10.1016/J.COMCOM.2018.02.011. URL: https://doi.org/10.1016/j.comcom.2018.02.011.

[67] 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.

[68] Naveed Ahmed Khan et al. “Achieving energy efficiency through load balancing: A comparison through formal verification of two WSN routing protocols”. In: *2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE. 2016, pp. 350–354.

[69] Harith Kharrufa, Hayder AA Al-Kashoash, and Andrew H Kemp. “RPL-based routing protocols in IoT applications: A Review”. In: *IEEE Sensors Journal* 19.15 (2019), pp. 5952–5967.

[70] Hyung-Sin Kim et al. “Challenging the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL): A Survey”. In: *IEEE Commun. Surv. Tutorials* 19.4 (2017), pp. 2502–2525. DOI: `10.1109/COMST.2017.2751617`. URL: `https://doi.org/10.1109/COMST.2017.2751617`.

[71] 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.

[72] Hideharu Kojima and Naoto Yanai. “A State Space Reduction Method for Model Checking of Wireless Multi-Hop Network Routing Protocols Focusing on Topologies”. In: *Seventh International Symposium on Computing and Networking Workshops, CANDAR 2019 Workshops, Nagasaki, Japan, November 26-29, 2019*. IEEE, 2019, pp. 14–20. DOI: `10.1109/CANDARW.2019.00010`. URL: `https://doi.org/10.1109/CANDARW.2019.00010`.

[73] Igor Konnov. URL: `https://protocols-made-fun.com/consensus/matterlabs/quint/specification/modelchecking/2024/07/29/chonkybft.html`.

[74] Moez Krichen. “A survey on formal verification and validation techniques for internet of things”. In: *Applied Sciences* 13.14 (2023), p. 8122.

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

[76] Satish Kumar and Chetana Asbe. “Development of Sustainable Message Forwarding Scheme for WSN by Data Fusion Multipath Routing Method”. In: *2023 IEEE 4th Annual Flagship India Council International Subsections Conference (INDISCON)*. 2023, pp. 1–7. DOI: `10.1109/INDISCON58499.2023.10270418`.

[77] 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.

[78] Marta Z. Kwiatkowska et al. “Performance analysis of probabilistic timed automata using digital clocks”. In: *Formal Methods Syst. Des.* 29.1 (2006), pp. 33–78. DOI: `10.1007/S10703-006-0005-2`.

[79] Hanane Lamaazi and Nabil Benamar. “A comprehensive survey on enhancements and limitations of the RPL protocol: A focus on the objective function”. In: *Ad Hoc Networks* 96 (2020). DOI: `10.1016/J.ADHOC.2019.102001`. URL: `https://doi.org/10.1016/j.adhoc.2019.102001`.

[80] Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002. ISBN: 0-3211-4306-X. URL: `http://research.microsoft.com/users/lamport/tla/book.html`.

[81] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. "UPPAAL in a Nutshell". In: *Int. J. Softw. Tools Technol. Transf.* 1.1-2 (1997), pp. 134–152. DOI: `10.1007/S100090050010`. URL: `https://doi.org/10.1007/s100090050010`.

[82] Fateh Latreche and Faiza Belala. "Timed CTL checking of time critical cloud applications using timed bigraphs". In: *Int. J. Crit. Comput. Based Syst.* 9.4 (2019), pp. 379–406. DOI: `10.1504/IJCCBS.2019.106818`.

[83] Axel Legay, Benoît Delahaye, and Saddek Bensalem. "Statistical Model Checking: An Overview". In: *Runtime Verification*. Ed. by Howard Barringer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 122–135. ISBN: 978-3-642-16612-9.

[84] P Levis et al. *The Trickle Algorithm*. RFC 6206. Mar. 2011. DOI: `10.17487/RFC6206`. URL: `https://www.rfc-editor.org/info/rfc6206`.

[85] Philip Alexander Levis et al. "TOSSIM: accurate and scalable simulation of entire tinyOS applications". In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, Los Angeles, California, USA, November 5-7, 2003*. Ed. by Ian F. Akyildiz et al. ACM, 2003, pp. 126–137. DOI: `10.1145/958491.958506`. URL: `https://doi.org/10.1145/958491.958506`.

[86] Yahui Li et al. "A Survey on Network Verification and Testing With Formal Methods: Approaches and Challenges". In: *IEEE Commun. Surv. Tutorials* 21.1 (2019), pp. 940–969. DOI: `10.1109/COMST.2018.2868050`. URL: `https://doi.org/10.1109/COMST.2018.2868050`.

[87] 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).

[88] Anfeng Liu, Peng-Hui Zhang, and Zhi-Gang Chen. "Theoretical analysis of the lifetime and energy hole in cluster based wireless sensor networks". In: *J. Parallel Distributed Comput.* 71.10 (2011), pp. 1327–1355. DOI: `10.1016/J.JPDC.2011.05.003`. URL: `https://doi.org/10.1016/j.jpdc.2011.05.003`.

[89] 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.

[90] Smitesh Mangelkar, Sudhir N Dhage, and Anant V Nimkar. “A comparative study on RPL attacks and security solutions”. In: *2017 International Conference on Intelligent Computing and Control (I2C2)*. IEEE. 2017, pp. 1–6.

[91] MarketsandMarkets™ INC. *Smart Sensors Market*. `https://www.marketsandmarkets.com/PressReleases/smart-sensor.asp`, Last accessed on 2024-05-10. 2024.

[92] José Antonio Mateo et al. “Probabilistic Model Checking: One Step Forward in Wireless Sensor Networks Simulation”. In: *Int. J. Distributed Sens. Networks* 11 (2015), 285396:1–285396:11. DOI: `10.1155/2015/285396`. URL: `https://doi.org/10.1155/2015/285396`.

[93] Maria Maximova, Holger Giese, and Christian Krause. “Probabilistic timed graph transformation systems”. In: *J. Log. Algebraic Methods Program.* 101 (2018), pp. 110–131. DOI: `10.1016/J.JLAMP.2018.09.003`.

[94] Anthéa Mayzaud, Remi Badonnel, and Isabelle Chrisment. “A Taxonomy of Attacks in RPL-based Internet of Things”. In: *Int. J. Netw. Secur.* 18.3 (2016), pp. 459–473. URL: `http://ijns.jalaxy.com.tw/contents/ijns-v18-n3/ijns-2016-v18-n3-p459-473.pdf`.

[95] 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.

[96] Robin Milner. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science. Springer, 1980. ISBN: 3-540-10235-3. DOI: `10.1007/3-540-10235-3`. URL: `https://doi.org/10.1007/3-540-10235-3`.

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

[98] 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`.

[99] Alekha Kumar Mishra et al. “Hybrid Mode of Operations for RPL in IoT: A Systematic Survey”. In: *IEEE Trans. Netw. Serv. Manag.* 19.3 (2022), pp. 3574–3586. DOI: `10.1109/TNSM.2022.3159241`. URL: `https://doi.org/10.1109/TNSM.2022.3159241`.

[100] 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.

[101] Chaib Mostefa et al. “Formal Validation of ADR Protocol in LoraWan Network Using Event-b”. In: *2023 7th International Conference on Computer, Software and Modeling (ICCSM)*. IEEE. 2023, pp. 11–15.

[102] M. Saqib Nawaz et al. “A Survey on Theorem Provers in Formal Methods”. In: *CoRR* abs/1912.03028 (2019). arXiv: `1912.03028`. URL: `http://arxiv.org/abs/1912.03028`.

[103] Anand Nayyar and Rajeshwar Singh. “A comprehensive review of simulation tools for wireless sensor networks (WSNs)”. In: *Journal of Wireless Networking and Communications* 5.1 (2015), pp. 19–47.

[104] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Vol. 2283. Lecture Notes in Computer Science. Springer, 2002. ISBN: 3-540-43376-7. DOI: `10.1007/3-540-45949-9`. URL: `https://doi.org/10.1007/3-540-45949-9`.

[105] Gethin Norman et al. “Model checking the probabilistic pi-calculus”. In: *Fourth International Conference on the Quantitative Evaluation of Systems (QEST 2007)*. IEEE. 2007, pp. 169–178.

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

[107] George Oikonomou et al. “The Contiki-NG open source operating system for next generation IoT devices”. In: *SoftwareX* 18 (2022), p. 101089. ISSN: 2352-7110. DOI: `https://doi.org/10.1016/j.softx.2022.101089`.

[108] Fernando Ojeda et al. “On wireless sensor network models: A cross-layer systematic review”. In: *Journal of Sensor and Actuator Networks* 12.4 (2023), p. 50.

[109] ZZ Oo and S Phyu. “Greenhouse environment monitoring and controlling system based on IoT technology”. In: *AIP Conference Proceedings*. Vol. 2339. 1. AIP Publishing. 2021.

[110] Agnieszka Paszkowska and Konrad Iwanicki. “On Designing Provably Correct DODAG Formation Criteria for the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL)”. In: *14th International Conference on Distributed Computing in Sensor Systems, DCOSS 2018, New York, NY, USA, June 18-20, 2018*. IEEE, 2018, pp. 43–52. DOI: `10.1109/DCOSS.2018.00014`. URL: `https://doi.org/10.1109/DCOSS.2018.00014`.

[111] Eloi Pereira et al. “BigActors: a model for structure-aware computation”. In: *ACM/IEEE 4th International Conference on Cyber-Physical Systems (with CPS Week 2013), ICCPS ’13, Philadelphia, PA, USA, April 8-11, 2013*. Ed. by Chenyang Lu, P. R. Kumar, and Radu Stoleru. ACM, 2013, pp. 199–208. DOI: `10.1145/2502524.2502551`. URL: `https://doi.org/10.1145/2502524.2502551`.

[112] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. *RFC3561: Ad hoc on-demand distance vector (AODV) routing*. 2003.

[113] Gian Perrone, Søren Debois, and Thomas T. Hildebrandt. “A model checker for Bigraphs”. In: *Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012*. Ed. by Sascha Ossowski and Paola Lecca. ACM, 2012, pp. 1320–1325. DOI: 10.1145/2245276.2231985. URL: https://doi.org/10.1145/2245276.2231985.

[114] Sophia Petridou, Stylianos Basagiannis, and Manos Roumeliotis. “Survivability analysis using probabilistic model checking: A study on wireless sensor networks”. In: *IEEE systems journal* 7.1 (2012), pp. 4–12.

[115] Yue Qiu and Maode Ma. “An authentication and key establishment scheme to enhance security for M2M in 6LoWPANs”. In: *IEEE International Conference on Communication, ICC 2015, London, United Kingdom, June 8-12, 2015, Workshop Proceedings*. IEEE, 2015, pp. 2671–2676. DOI: 10.1109/ICCW.2015.7247582. URL: https://doi.org/10.1109/ICCW.2015.7247582.

[116] Shalli Rani, Syed H. Ahmed, and SpringerLink (Online service). *Multi-hop routing in wireless sensor networks: an overview, taxonomy, and research challenges*. English. Singapore: Springer, 2015. ISBN: 9812877304.

[117] Adnan Rashid, Osman Hasan, and Kashif Saghar. “Formal analysis of a ZigBee-based routing protocol for smart grids using UPPAAL”. In: *2015 12th International Conference on High-capacity Optical Networks and Enabling/Emerging Technologies (HONET)*. IEEE. 2015, pp. 1–5.

[118] Wolfgang Reisig. *Petri nets: an introduction*. Vol. 4. Springer Science & Business Media, 2012.

[119] Wolfgang Reisig and Grzegorz Rozenberg. *Lectures on petri nets i: basic models: advances in petri nets*. Springer Science & Business Media, 1998.

[120] 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.

[121] *Routing Information Protocol*. RFC 1058. June 1988. DOI: 10.17487/RFC1058. URL: https://www.rfc-editor.org/info/rfc1058.

[122] Neda Saeedloei and Gopal Gupta. “Timed π-Calculus”. In: *Trustworthy Global Computing - 8th International Symposium, TGC Argentina, August 30-31, 2013, Revised Selected Papers*. Ed. by Martín Abadi and Alberto Lluch-Lafuente. Vol. 8358. Lecture Notes in Computer Science. Springer, pp. 119–135. DOI: 10.1007/978-3-319-05119-2_8.

[123] Hamza Sahli, Faiza Belala, and Chafia Bouanaka. “A BRS-Based Approach to Model and Verify Cloud Systems Elasticity”. In: *Procedia Computer Science* 68 (2015). 1st International Conference on Cloud Forward: From Distributed to Complete Computing, pp. 29–41. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.09.221. URL: https://www.sciencedirect.com/science/article/pii/S1877050915030665.

[124] 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.

[125] Zohra Sbaï and Mohamed Escheikh. “Model Checking Techniques for Verification of an Encryption Scheme for Wireless Sensor Networks”. In: *CoRR* abs/1305.4247 (2013). arXiv: 1305.4247. URL: http://arxiv.org/abs/1305.4247.

[126] 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.

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

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

[129] 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.

[130] 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.

[131] Elhadi M. Shakshuki, Haroon Malik, and Tarek R. Sheltami. “A comparative study on simulation vs. real time deployment in wireless sensor networks”. In: *J. Syst. Softw.* 84.1 (2011), pp. 45–54. DOI: 10.1016/J.JSS.2010.08.028. URL: https://doi.org/10.1016/j.jss.2010.08.028.

[132] Sharad, Er Navpreet Kaur, and Inderdeep Kaur Aulakh. “Evaluation and implementation of cluster head selection in WSN using Contiki/Cooja simulator”. In: *Journal of Statistics and Management Systems* 23.2 (2020), pp. 407–418.

[133] Richa Sharma, Vasudha Vashisht, and Umang Singh. “Modelling and simulation frameworks for wireless sensor networks: a comparative study”. In: *IET Wireless Sensor Systems* 10.5 (2020), pp. 181–197.

[134] Richa Sharma, Vasudha Vashisht, and Umang Singh. “Modelling and simulation frameworks for wireless sensor networks: a comparative study”. In: *IET Wirel. Sens. Syst.* 10.5 (2020), pp. 181–197. DOI: 10.1049/IET-WSS.2020.0046. URL: https://doi.org/10.1049/iet-wss.2020.0046.

[135] Dr Kshitij Shinghal et al. “Intelligent humidity sensor for-wireless sensor network agricultural application”. In: *International Journal of Wireless & Mobile Networks (IJWMN)* 3.1 (2011), pp. 118–128.

[136] Ahmed Sobeih et al. “J-Sim: A Simulation Environment for Wireless Sensor Networks”. In: *Proceedings 38th Annual Simulation Symposium (ANSS-38 2005), 4-6 April 2005, San Diego, CA, USA*. IEEE Computer Society, 2005, pp. 175–187. DOI: 10.1109/ANSS.2005.27. URL: https://doi.org/10.1109/ANSS.2005.27.

[137] Peng Sun, Azzedine Boukerche, and Yanjie Tao. “Theoretical Analysis of the Area Coverage in a UAV-based Wireless Sensor Network”. In: *13th International Conference on Distributed Computing in Sensor Systems, DCOSS 2017, Ottawa, ON, Canada, June 5-7, 2017*. IEEE, 2017, pp. 117–120. DOI: 10.1109/DCOSS.2017.18. URL: https://doi.org/10.1109/DCOSS.2017.18.

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

[139] Mamoona Tariq and Kashif Saghar. “Evaluation of a sensor network node communication using formal verification”. In: *2015 12th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE. 2015, pp. 268–271.

[140] 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.

[141] Sandeep A Thorat, PJ Kulkarni, and Shital V Yadav. “Formal verification of opportunistic routing protocol using SPIN model checker”. In: *2017 international conference on energy, communication, data analytics and soft computing (ICECDS)*. IEEE. 2017, pp. 2717–2722.

[142] 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.

[143] Llanos Tobarra et al. “Model checking wireless sensor network security protocols: TinySec + LEAP + TinyPK”. In: *Telecommun. Syst.* 40.3-4 (2009), pp. 91–99. DOI: 10.1007/S11235-008-9131-Z. URL: https://doi.org/10.1007/s11235-008-9131-z.

[144] Ivana Tomić and Julie A McCann. “A survey of potential security issues in existing wireless sensor network protocols”. In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1910–1923.

[145] 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.

[146] Kazunori Ueda. “LMNtal as a hierarchical logic programming language”. In: *Theoretical Computer Science* 410.46 (2009), pp. 4784–4800.

[147] Abhishek Verma and Virender Ranga. “Security of RPL based 6LoWPAN Networks in the Internet of Things: A Review”. In: *IEEE Sensors Journal* 20.11 (2020), pp. 5666–5690.

[148] 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).

[149] Dargie Waltengus and Poellabauer Christian. *Fundamentals of wireless sensor networks: theory and practice*. 2010.

[150] Na Wang, Jiacun Wang, and Xuemin Chen. “A trust-based formal model for fault detection in wireless sensor networks”. In: *Sensors* 19.8 (2019), p. 1916.

[151] MP Webster et al. “Formal verification of synchronisation, gossip and environmental effects for critical IoT systems”. In: (2018).

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

[153] 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.

[154] Wanling Xie, Huibiao Zhu, and Qiwen Xu. “BigrTiMo-A Process Algebra for Structure-Aware Mobile Systems”. In: *22nd International Conference on Engineering of Complex Computer Systems, ICECCS, Fukuoka, Japan, November 5-8, 2017*. IEEE Computer Society, pp. 50–59. DOI: 10.1109/ICECCS.2017.13.

[155] Zhu Xin-feng et al. “Methods to tackle state explosion problem in model checking”. In: *2009 Third International Symposium on Intelligent Information Technology Application*. Vol. 2. IEEE. 2009, pp. 329–331.

[156] 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.

[157] Kaiyu Yang and Jia Deng. "Learning to Prove Theorems via Interacting with Proof Assistants". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6984–6994. URL: `http://proceedings.mlr.press/v97/yang19a.html`.

[158] Kok-Kiong Yap, Vikram Srinivasan, and Mehul Motani. "MAX: human-centric search of the physical world". In: *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, SenSys 2005, San Diego, California, USA, November 2-4, 2005*. Ed. by Jason Redi, Hari Balakrishnan, and Feng Zhao. ACM, 2005, pp. 166–179. DOI: `10.1145/1098918.1098937`. URL: `https://doi.org/10.1145/1098918.1098937`.

[159] Qiang Ye and Mike H. MacGregor. "Using simulation to test formally verified protocols in complex environments". In: *Math. Comput. Model.* 53.3-4 (2011), pp. 538–551. DOI: `10.1016/J.MCM.2010.03.039`. URL: `https://doi.org/10.1016/j.mcm.2010.03.039`.

[160] Wang Yi. "CCS + Time = An Interleaving Model for Real Time Systems". In: *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Spain, July 8-12, 1991, Proceedings*. Ed. by Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo. Vol. 510. Lecture Notes in Computer Science. Springer, pp. 217–228. DOI: `10.1007/3-540-54233-7\_136`.

[161] Lian Yu et al. "Modeling and Analysis of Mobile Cloud Computing Based on Bigraph Theory". In: *2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud United Kingdom, April 8-11, 2014*. IEEE Computer Society, pp. 67–76. DOI: `10.1109/MOBILECLOUD.2014.11`.

[162] Rachid Zagrouba and Amine Kardi. "Comparative study of energy efficient routing techniques in wireless sensor networks". In: *Information* 12.1 (2021), p. 42.

[163] Azam Zavvari et al. "Theoretical analysis of RFID security protocols". In: *2014 IEEE International Conference on Industrial Engineering and Engineering Management, IEEM 2014, Selangor Darul Ehsan, Malaysia, December 9-12, 2014*. IEEE, 2014, pp. 302–306. DOI: `10.1109/IEEM.2014.7058648`. URL: `https://doi.org/10.1109/IEEM.2014.7058648`.

[164] Weiyu Zhong et al. "Byzantine fault-tolerant consensus algorithms: A survey". In: *Electronics* 12.18 (2023), p. 3801.

[165] Siham Zroug et al. "A hierarchical formal method for performance evaluation of WSNs protocol". In: *Computing* 103.6 (2021), pp. 1183–1208.

[166] Wlodek M Zuberek. “Timed Petri nets definitions, properties, and applications”. In: *Microelectronics Reliability* 31.4 (1991), pp. 627–644.