Faculty of Engineering and Science
University of Agder

Master's Thesis

Modeling and Verifying the OLSR Protocol Using Uppaal

Author:
Mojgan Kamali

Supervisors:
Prof. Andreas Prinz
Dr. Peter Höfner

Department of Information and Communication Technology (ICT)

June 2014
Grimstad, Norway

University of Agder
Faculty of Engineering and Science

To my mom and dad

Abstract

Wireless Mesh Networks (WMNs) are a popular technology due to their flexibility and self-organizing nature that provide support for broadband communication. They are used in a wide range of application areas, such as public transportation, tunnels, real time racing car telemetry and emergency response communication. Route finding and maintenance, two important factors determining the performance of such networks, are provided using routing algorithms. The Optimized Link State Routing (OLSR) protocol is an example of such algorithms which is used in this study.

One issue about this protocol is that its specification is in English that may cause ambiguities or different interpretations. The *first contribution* of this project is the development of a formal and unambiguous model of OLSR and its main functionalities using timed automata as our formal specification language. The *second contribution* of the project is a precise analysis of OLSR using the model checker Uppaal. By a careful automated analysis with Uppaal, the project shows a complementary approach to classical techniques, such as test-bed experiments and simulation.

One *overall goal* of this study is the demonstration that automated, formal and rigorous analysis of real-world protocols is possible and can be achieved in a rather short period of time. Our model covers all core components of OLSR and abstracts from the optional features. At the moment, the project analyses fundamental behavior such as packet delivery; the model guarantees that a packet which is injected into a network is finally delivered at the destination. Moreover, the study verifies that nodes in the network can find shortest paths to other nodes.

Acknowledgments

This master thesis has been written at the Department of ICT, University of Agder. A number of people deserve thanks for their help and support. In this acknowledgement, it is therefore my greatest pleasure to express my gratitude to all of them.

First of all, I wish to thank my supervisors Prof. Andreas Prinz from ICT department at University of Agder, Norway, and Dr. Peter Höfner from NICTA, Australia, for their excellent advice and support, their valuable reviews of this thesis, for providing constructive comments that improved its quality, for the countless scientific discussions we have had, and for always being there to help.

Furthermore, I would like to express my special gratitude and thanks to my family and friends for their continuous encouragement and support throughout these two years of study. My thanks and appreciations go especially to my brothers, Ehsan and Morteza, and my dear sister, Maryam, who always have been constant source of inspiration. I would like to express my deep sense of gratitude to my dearest parents, Nahid and Ali, for giving birth to me at the first place, their love and supporting me spiritually throughout my life. I will be grateful forever for your support. This thesis is dedicated to you.

Mojgan Kamali
June 2014

Acronyms

AODV	Ad hoc On-Demand Distance Vector
B.A.T.M.A.N.	Better Approach To Mobile Adhoc Networking
CAGR	Compound Annual Growth Rate
CTL	Computational Tree Logic
DSDV	Destination-Sequenced Distance Vector
DSL	Digital Subscriber Line
DYMO	Dynamic MANET On-demand
IETF	Internet Engineering Task Force
LTL	Linear Temporal Logic
MAC	Media Access Control
MANET	Mobile Ad-hoc Network
MPR	Multipoint Relay
OLSR	Optimized Link State Routing
PC	Personal Computer
RFC	Request for Comments
SMC	Statistical Model Checking
TC	Topology Control
TTL	Time To Live
WLAN	Wireless Local Area Network
WMC	Wireless Mesh Client
WMN	Wireless Mesh Network
WMR	Wireless Mesh Router

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless Mesh Networks (WMNs) have gained popularity and are increasingly applied in a wide range of application areas, including communication, emergency response networks, intelligent transportation systems, mining, outdoor enterprises and wireless video surveillance [6]. They are self-organizing wireless multi-hop networks which are able to provide support for broadband communication without relying on a wired infrastructure [7]. As a consequence, they bear the benefit of rapid and low-cost network deployment. "WMNs can be considered a superset of Mobile Ad hoc Networks (MANETs)" [7, p.4], where a network contains mobile end user devices such as smartphones or laptops. WMNs might also consist stationary infrastructure devices known as mesh routers in contrast to MANETs.

In 2008, the global WMN Market was valued at around \$3.98Bn with over 1,443M units (wireless mesh routers) shipped, with a forecast Compound Annual Growth Rate (CAGR) of approximately 2.7% [8]. Future growth areas for WMNs include application to the Mobile Backhaul Infrastructure market. Defined as the network used to transport call traffic between the cell tower and mobile switching centre, this market/ application area had global revenues of approximately \$7.117Bn in 2008, with a forecast CAGR (2008-2015) of 19.6% and forecast revenues of \$24Bn in 2014 [9]. This illustrates the importance of WMNs in today's information and communication technology. At the moment, the WMN marked is comprised of the following key applications areas:

- Municipal services (public access internet, public safety and security, etc).
- Transportation (bus, , trains,etc).
- Enterprise networks (business applications including data, voice, and video).
- Emergency (detection and response communication).

Most of the applications require reliable real-time communication, which is increasingly multimedia in nature. An example deployment of a WMN is depicted in Fig. 1.1. It shows a scenario, where sensors, which also act as mesh routers[1], are deployed in a

[1] Mesh routers are devices applied for routing data packets from one place to another.

Fig. 1.1. Detecting fire in a forest using WMNs [1].

forest to detect bush fires. As soon as a sensor detects a heat (a fire), some authority has to be informed. Since usually the transmission range of the routers is limited, one has to make use of intermediate routers to reach the destination. However, if a bush fire is in place, it might be that some of the routers (sensors) fail—for example a sensor could be damaged or burnt. Due to this, messages indicating the fire outbreak must be sent via an existing and working multi hop path. This indicates that routing protocols must be able to react rapidly to change topologies.

It has been pointed out by end users that current wireless mesh solutions do not consistently meet expectations. For example, complaints voiced by Australian Police and other Australian municipal public safety service providers in private conversations cite difficulty in reliably transmitting real-time multi-media content using current WMN technology across a variety of application scenarios (e.g., fluctuating link quality, changing position of nodes and moving vehicles) [10]. It is the belief of some researchers that the failure of existing WMN systems are to a large extent due to limitations of current network protocols and their inability to tailor and adapt their operation to the very different and dynamic deployment environments of WMNs. In this study, we are focussing on new adaptive wireless network protocols that are supposed to overcome the limitations of current systems. As a general guideline, it can be stated that limitations, shortcomings or problems in the routing protocol immediately decrease the performance of the entire network.

A routing protocol enables node communication in a network by disseminating information that enables the nodes to select routes. By this, nodes are able to sent data packets to arbitrary destinations in the network. As a consequence, one of the key factors determining the performance and reliability of WMNs is the routing protocol [7]. The correctness and performance of the routing protocol as a crucial factor for the reliable behavior of a network must be checked.

1.1 Problem Statement

Traditionally, common methods used to evaluate and validate network protocols are test-bed experiments and simulation. However, these are valid and important techniques used for protocol evaluation, especially for quantitative performance evaluation, they have some shortages with respect to the evaluation of basic protocol correctness properties [7]. In addition, experimental evaluation is expensive, time-consuming and resource intensive. As a consequence, protocol limitations and errors are still found many years after the definition and standardization.

Another problem of routing protocols is that they are usually specified by English prose. Although this makes it easy to understand, it is well known that textual description contain ambiguities, contradictions and lack often some details. As a consequence, this might yield to different interpretations of one specification, yield different implementations. In the worst case implementations of the same routing protocol are even incompatible—which should not be the case; in WMNs it should not matter which implementation single mesh routers use; as long as they use implementations for the same routing protocol. These problems open a new horizon for researchers to think about how to overcome these issues.

To underpin our belief and to illustrate that reasonably rich Internet Engineering Task Force (IETF) protocols can be formally modeled, we focus on the Optimized Link State Routing (OLSR) protocol. This protocol has been identified as the standard proactive ad hoc routing protocol by the IETF MANET working group[2]. It is a proactive routing protocol particularly designed for WMNs. One *overall goal* of this study is the demonstration that automated, formal and rigorous analysis of real-world protocols is possible and can be achieved in a rather short period of time. Our formal model covers all core components of OLSR and abstracts from the optional features. At the moment, the project analyses fundamental behavior such as packet delivery; it guarantees that a packet which is injected into a network finally is delivered at the destination. Moreover, the study verifies that nodes in the network can find shortest paths to other nodes.

1.2 Approach

One approach to address this problem, ambiguities of a specification, is using formal methods in general and model checking in particular. Formal methods provide valuable tools for designing, evaluating and verifying of WMN routing protocols which complement some other experiments, such as test-bed experiments and simulation. These methods have a great potential to improve the correctness and precision of design and development and they produce reliable results. Formal methods allow formal specification of routing protocols and verification of their desired behavior by applying mathematics and logics. By applying these methods, stronger and more general assurance about protocol behavior and properties can be achieved. Model checkers allow to express properties of the system using temporal logic formula. They are able to "validate key correctness properties in finite representation of a formal system model" [11, p.1]. The Uppaal model checker, a tool for specification and verification of real-time systems, particularly is used in this study to explore the behavior of the OLSR protocol. In this project, we derive a formal specification based on timed automata.

The *first contribution* of this project is developing a formal and unambiguous model of OLSR and its main functionalities using timed automata as our formal

[2] `http://datatracker.ietf.org/wg/manet/charter/`

specification language. Timed automata are finite automata extended by clocks which provide the right level of abstraction and allow the specification of all core aspects of OLSR. The extension also features the modeling of simple data structures, such as routing tables, as needed for an accurate model of OLSR. Moreover, timed automata are used as input for the model checker Uppaal for analysis of OLSR.

The *second contribution* of the project is a precise analysis of OLSR using the model checker Uppaal. The analyze can only be performed after the development of a formal and unambiguous description of OLSR. We analyze the protocol with respect to basic requirements such as packet delivery. By a careful automated analysis with Uppaal, the project shows a complementary approach to classical techniques.

We apply Uppaal for the following reasons:

1. Two synchronization mechanisms provided by Uppaal, binary and broadcast synchronization which will be discussed in section 3.2, fit perfectly to model unicast and broadcast communications in WMNs.

2. Common data structure, such as structs and arrays are provided by Uppaal and also C-like programming language defines updates on these data structures [12].

3. Since OLSR is a proactive protocol, it highly depends on time. Uppaal provides mechanism for considering time variables.

The use of Uppaal lies partly in the cast that this tool has been used in numerous case studies and is well established; moreover since it is on the market for over 20 years now, the system can be considered “trustworthy”. In addition, Uppaal is a well established tool for verifying properties of the system. However, it has the problem of model checkers, state space explosion.

1.3 Importance of Topic

WMNs are applied in a wide range of applications and they can be established in physical phenomena for different purposes such as emergency situations, battlefield surveillance and so on. Such networks are very common and the demand for using these networks is undeniable. A rough estimate of the financial and economic importance of WMNs has been given in the first part of this study (introduction). The numbers show that the market for WMNs is a multi-billion market, which will even increase in the future.

Correctness, reliability and confidence of such networks and their routing protocols are very fundamental and vital due to their safety-critical applications. For instance, establishing mesh networks to monitor volcanoes aiming at preventing disasters could be one of the most important usages. Fig. 1.2 represents the attempt of scientists to install five mobile stations to detect an event. Incorrect communication may cause complications ranging from financial detriments to health issues. Moreover, failure of expected functions might cause catastrophic results.

Providing an understandable and correct model of the OLSR protocol can be considered as a step towards better and high assured protocols. In addition, verifying the packet delivery mechanism of OLSR and its correctness are two other important concerns. As OLSR is still under development, we provide valuable feedback[3] to the developers, who might take our findings into account when generating the final specification of OLSR.

Fig. 1.2. Monitoring a volcano using WMNs [2].

1.4 Thesis Organization

This thesis is organized as follows. In chapter 2, we give an overview of WMNs, and routing protocols used in these networks, particularly the OLSR protocol. We describe formal methods and Uppaal which is the tool used in this study, in chapter 3. We review related work in chapter 4. In chapter 5, we explain the Uppaal model of OLSR, which is based on RFC 3626 [13]. Our experiments results are presented in chapter 6. Finally, we summarize our work and propose future directions in chapter 7.

[3] Our feedback will be mentioned in chapter 7.

Chapter 2

Wireless Mesh Networks

Wireless Mesh Networks have appeared as a promising technology to meet next generation networks challenges such as flexibility, adaptability and having reconfigurable architecture by proposing cost-effective solutions to the service providers. They are communications networks made up of several nodes organized in mesh topologies. WMNs comprise Wireless Mesh Clients (WMCs) and Wireless Mesh Routers (WMRs) and are based on wireless multi-hop transmission. In these networks, without the need for running any routing feature, clients associate to a WMR [4]. Fig. 2.1 shows a sample of WMN.

In this chapter, we give an overview of WMNs, their applications and challenges, routing protocols used in these networks, and finally we describe the OLSR protocol which is our focus in this study.

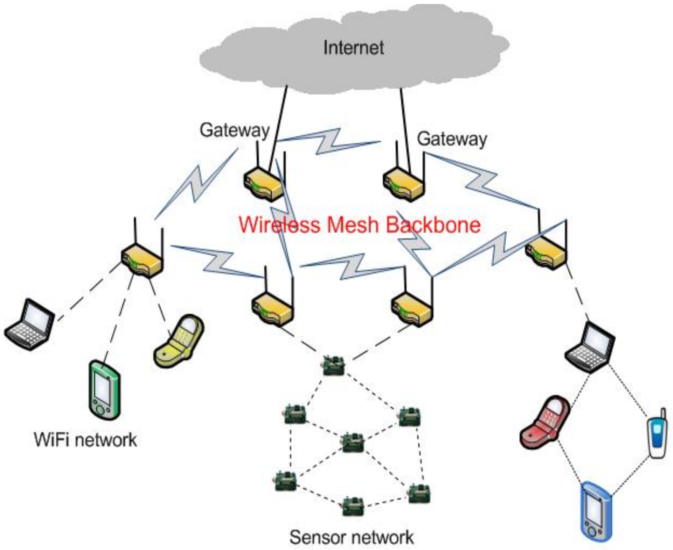

Fig. 2.1. The architecture of WMNs [3].

2.1 WMNs Architecture

WMNs contain two different kind of nodes, mesh clients and mesh routers. Wireless mesh routers support mesh networking by routing capability for gateway functions together with additional routing functions. A mesh router is usually equipped with other wireless interfaces to enhance the flexibility of mesh networking [4]. The WMNs architecture can be divided into three main categories based on the functionality of the nodes.[1]

2.1.1 Infrastructure/Backbone WMNs

A sample backbone WMN is indicated in Fig. 2.2 in which solid lines and dash represent wired and wireless links, respectively. This type of WMNs contains mesh routers organizing an infrastructure for clients that connect to them. Various types of radio technologies, such as IEEE 802.11, can be used to build the WMN infrastructure/backbone. A mesh of self-configuring [14] and self-healing [14] links can be formed via mesh routers, and these mesh routers can be connected to the Internet via gateway functionality. Ethernet links can be used to connect conventional clients with Ethernet interface to mesh routers. In case of having different radio technologies, clients have to communicate with the base stations that have Ethernet connections to mesh routers. This type of architecture, Infrastructure/Backbone WMNs, is the most commonly used type. For instance, infrastructure meshing can be used to build community and neighborhood networks. The mesh routers serve as access points for users inside the houses which can be placed on the roof of homes in a neighborhood. Typically, two types of radios, for backbone and for user communication, are used in the routers. Long-range communication approaches including directional antennas can be used to establish mesh backbone communication [4].

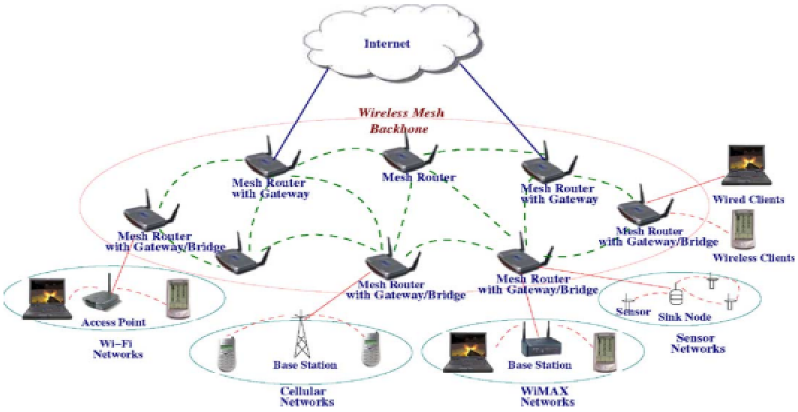

Fig. 2.2. Infrastructure/Backbone WMNs [4].

[1] Owing to applicability of OLSR in WMNs, we describe all different types of WMNs architectures.

2.1.2 Client WMNs

Peer-to-peer networks among client devices can be provided by client meshing. This type of architecture provides the situation where *a*) client nodes establish the actual network for performing configuration functionalities and routing, *b*) customers can access end user applications. As a consequence, these types of networks do not require a mesh router. The architecture is illustrated in Fig. 2.3. In Client WMNs, an injected packet traverse paths by hopping from node to node until the destination is reached. One type of radios is usually used to form client WMNs. In this architecture, the end-users have to perform some additional function, such as self configuration and routing. Therefore compared to infrastructure meshing, the requirements on end-user devices will be incremented [4].

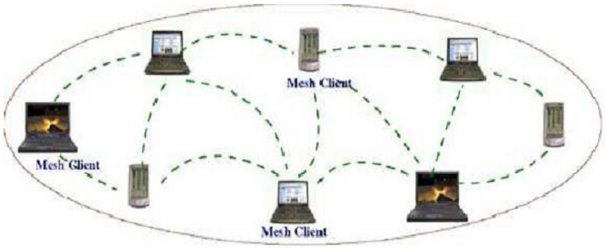

Fig. 2.3. Client WMNs [4].

2.1.3 Hybrid WMNs

This architecture is the combination of two previous discussed architecture, infrastructure and client meshing. Fig. 2.4 demonstrates this architecture. The network can be accessed via mesh routers. Connectivity to other networks such as the Internet, cellular, Wi-Fi and sensor networks can be provided by the infrastructure, and better coverage and connectivity inside the WMN is enabled via clients routing capabilities [4].

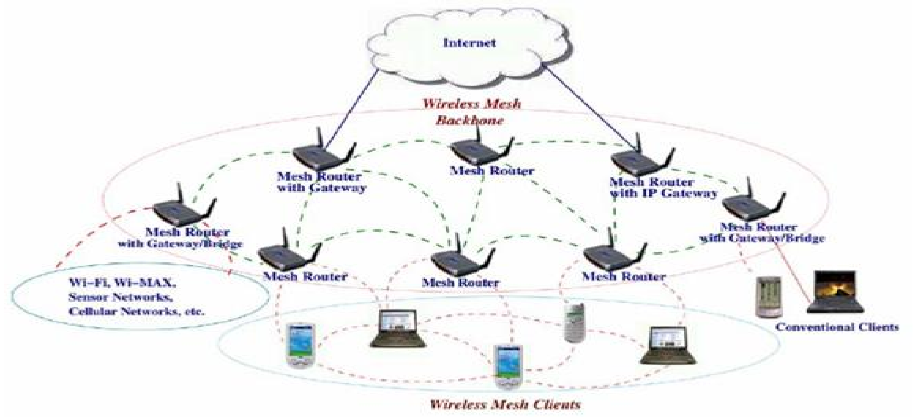
Internet
Wireless Mesh Backbone
Mesh Router with Gateway
Mesh Router
Mesh Router with IP Gateway
Mesh Router with Gateway/Bridge
Mesh Router with Gateway/Bridge
Mesh Router
Mesh Router
Mesh Router
Wi-Fi, Wi-MAX, Sensor Networks, Cellular Networks, etc.
Conventional Clients
Wireless Mesh Clients

Fig. 2.4. Hybrid WMNs [4].

2.2 Applications of WMNs

The concept of peer-to-peer mesh topology with wireless communication along mesh routers is introduced by WMNs. This concept is assisting to overcome several challenges in today's WMNs deployment. Installation of extensive Ethernet cabling can be mentioned as an example of these challenge. Some instances of well suited deployment scenarios for WMNs are as follows:

- Enterprise networks (shopping centers, airports, and special events).
- Emergency responses (military, temporary installation, and disaster recovery).
- Municipalities (residential regions, downtown cores, parks and public safety).
- Transportation (railways, bus, etc).

According to recent studies in WMNs, these networks have been used in many different applications. The WMNs topology enables several alternative paths for a given pair of source and destination nodes, allowing quick reconfiguration of the path in case of path failure happening. Providing connectivity to mobile/static clients is possible by placing mesh routers elsewhere. In addition by incrementally adding mesh routers, the coverage area can be improved. These beneficial features of WMNs cause the growth in using WMNs in different applications as the following.

2.2.1 Home Networking

A network of home appliances such as television, video camera, personal computer, etc which are realized by Wireless Local Area Network (WLAN) technology is called broadband home networking. The obvious challenge in here is the place of access point in the home that might cause dead zones with no service coverage. For sure, one solution to overcome this problem is using multiple access points connected by

Ethernet cabling, but it leads to an increase in deployment cost and overhead. An impressive solution can be using mesh routers instead of all the access points and establishing mesh connectivity between those mesh routers. So, broadband connectivity between the home networking devices will be provided and only one single connection to the Internet is required via the gateway router. The dead zones can be removed by changing the location and number of mesh routers [4]. Fig. 2.5 depicts where mesh routers are used in one typical home network.

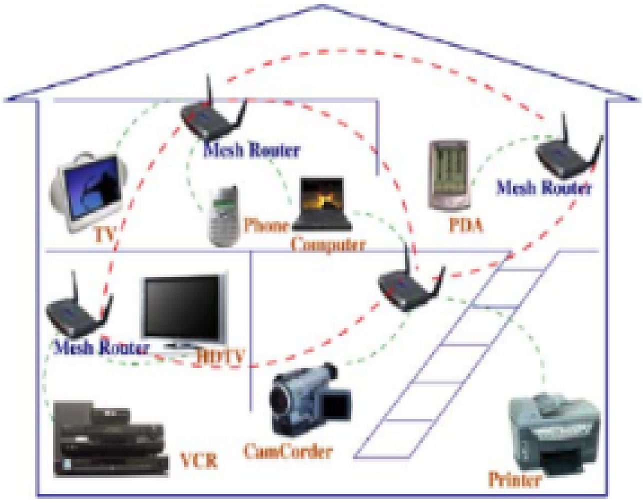

Fig. 2.5. Wireless mesh network-based Home networking [4].

2.2.2 Community Networking

Connecting the home network or Personal Computer (PC) to the Internet with a Digital Subscriber Line (DSL) modem or cable is the common way of establishing community networking which causes inefficient utilization of the network resources, since all the traffic goes through the Internet. Moreover, wireless connectivity do not always provide coverage outside the home. All of these problems have been solved by using WMNs which provides a cost effective way for sharing Internet among different homes as shown in Fig. 2.6. There are several advantages using such mesh networks. As a case in point, being faster and getting cost effective Internet access via distributed gateways. Another advantage that should be high lighted is that neighbors can use backup technology and the probability of losing information because of catastrophic disk failure decreases. Dissemination of relevant cached information to local community becomes much faster and easier. Mesh routers can be established on windows or rooftops easily and client devices can connect to them in a single hop [4].

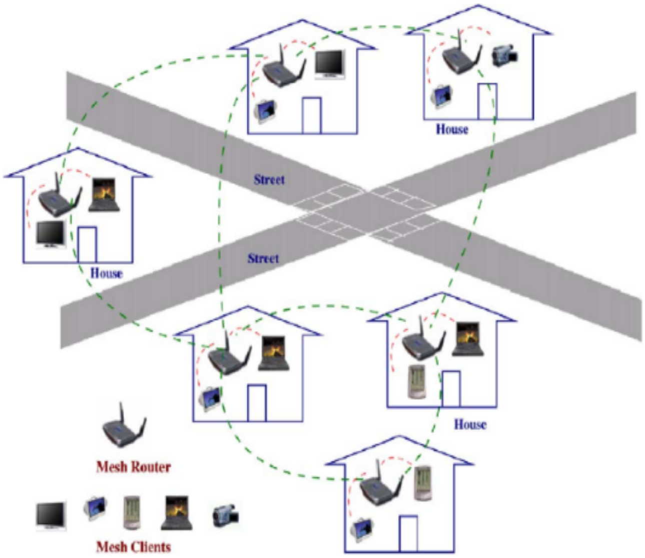

Fig. 2.6. Wireless mesh network-based Community Networking [4].

2.2.3 Disaster Management and Rescue Operations

WMNs can be applied where automatically network connectivity is needed, such as emergency operations and disaster management. While happening some disasters, such as earthquake, fire or flood, almost all the communication devices may stop working or get collapsed. As a consequence during the rescue operation, the rescue team vehicle can be equipped with mesh routers which allow rescue team members to communicate with each other, as shown in Fig. 2.7. Different mobile devices can access the network by providing different communication interfaces at mesh routers that make it possible for people to communicate when they are in critical situations. Since these networks can be formed in less time, the rescue operation can be more effective [4].

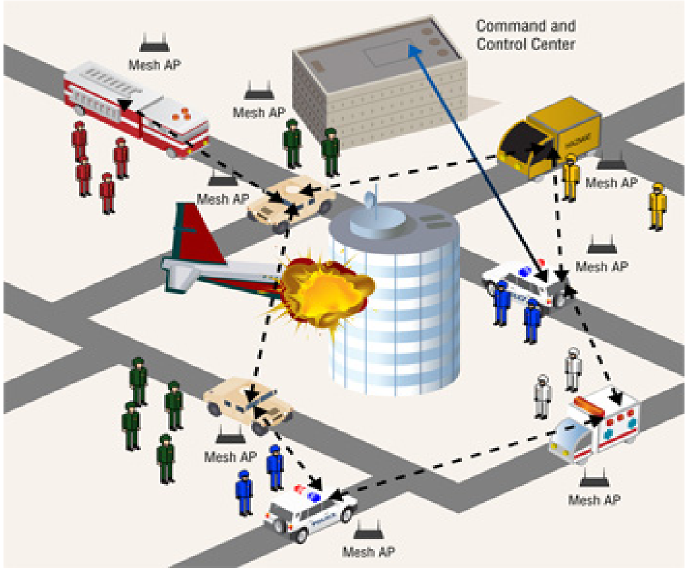

Fig. 2.7. Wireless mesh network based rescue operation [5].

2.3 WMNs Challenges

In spite of having attractive and strong features, WMNs have some challenges and options that need to be addressed. In this section, we explain some of these research challenges [4].

2.3.1 Performance Issues

The efficiency of any network is the most important and critical factor that must be considered before accepting and deploying at large scale for different applications. The issues that might affect the performance of WMNs are listed below:

2.3.1.1 Distributed MAC and Multi-hop Communication

The Media Access Control (MAC) function has to be accomplished in a distributed way since mesh networks have the decentralized nature. In addition, the MAC protocol for WMNs is pertained to more than one-hop connections. These requirements lead to highly challenging design of MAC functions.

2.3.1.2 Mesh Routing

In mesh networking, every node needs to share route information with other nodes. Mesh routing protocols assure this functionality. Some attempts have been initiated to adapt the ad-hoc routing protocols for WMNs. But since there are some open issues in the ad-hoc routing area with respect to important performance factors, such as scalability, fault tolerant, etc, these networks solutions are not appropriate for

WMNs. As a consequence, innovative solutions are necessary to resolve the above issues in WMNs.

2.3.2 Scalability

Scalability, reliability and robustness are significant and relevant issues that have to be addressed for enabling the operation of several embedded applications for WMNs. Due to the fact that multi-hop communication is common in WMNs, mesh networks have the typical scalability issues in multi-hop networking. Therefore when the size of the network increases, the end-to-end reliability drastically decreases and the network performance diminishes. For instance, routing protocols may loose connections or may be disabled to find a reliable routing path. So by using efficient and scalable routing protocols to transport data in a robust manner, it is possible to solve these existing challenges in WMNs [4].

2.4 WMN Routing Protocols

As discussed in section 2.3, the complexity of WMNs increases when having a large number of nodes. This turns network into a purpose from the point of reliability, security and manageability. Routing protocols for such networks are envisaged to provide some functions such as constructing and selecting routes, finding and responding to network topology changes, enabling management, increasing the network capacity and decreasing the packet delivery delays. A routing protocol determines the way of communication between routers and how they disseminate information to select routes from a source node to a destination node. It also provides the situation for transmitting packets through the network. It first shares the information between next neighbors and then to the whole network. Therefore, all routers can gain the knowledge of the network topology. Therefore, the routing protocol is one of the key factors of determining reliability and performance of such networks.

There are dozens of different proposed routing protocols in WMNs. Some of these protocols have been used for many years which have been standardized by IETF. These protocols are categorized into two main groups, table-driven or proactive protocols and on-demand or reactive protocols, which are discussed in this chapter.

2.4.1 Reactive Protocols

On-demand or reactive routing protocols establish routes only if needed, that means some mechanism is triggered as soon as a node needs to send data packets to another node. They find a route by sending route request packets through the network. The route discovery process is completed when a route from the source nodes to the destination node is determined. If one route has been found, it will be maintained by a process called route maintenance process until the destination is accessible and the route is desired. These type of protocols keep routes from the source to all active destinations and for each unknown destination, a route discovery process is required.

Therefore, the communication overhead will be decreased at the expense of delay because of route search. The Ad hoc On-Demand Distance Vector (AODV) routing protocol and the Dynamic MANET On-demand (DYMO also known as AODV(v2)) routing protocol are two examples of such protocols.

2.4.2 Proactive Protocols

Table-driven or proactive routing protocols establish routes in advance, i.e., these protocols assume that it is more efficient to determine routes regularly; as soon as a node tries to send data packets, the reasonable assumption (hope) is that the route needed has already been discovered. They maintain lists of available destinations and up-to-date routes to those destinations in the network by sending periodic routing information. These information keeps routing tables consistent. Due to having existing routes available, when a traffic packet arrives, transmission will occur with no delay. As a consequence, these types of protocols could be in principle more efficient with respect to time needed to deliver data packets. Some examples of proactive protocols are the Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.) protocol, the Optimized Link State Routing (OLSR) protocol and Destination-Sequence Distance Vector (DSDV) protocol.

2.4.3 Routing Protocol Challenges and Issues

Most of the protocols mentioned above, are specified by so called Request for Comments (RFC). A RFC is a document published by the Internet Engineering Task Force (IETF) or the Internet Society; in case of routing protocol the IETF is the responsible authority. In their final state RFCs are standards specifying the algorithm (routing protocol) in more or less detail. Unfortunately, all routing protocols for WMNs are described in English Prose. Of course a textual description of a protocol enables everybody to read (and understand) the standard. However, it is known to be hard to translate natural specifications into a formal model or executable code. This topic has been studied by researchers for quite some time. Lots of solutions have been proposed to overcome this problem, but they lead to clumsy, verbose and definitively unsuccessful versions at first stage [15]. One problem of the translation is that textual specifications often contain ambiguities, inaccuracies, or even contradictions.

It has also been confirmed in many case studies that RFCs written merely in a natural language contain ambiguities and contradictions. As a consequence, the various implementations, whether claimed to be RFC compliant or not, depart in various ways from the RFC [16]. Moreover, semi-informal reasoning is inadequate to ensure critical safety properties. We believe that formal specification languages and analysis techniques with rigorous mathematical underpinnings are nowadays able to capture the full syntax and semantics of reasonably rich IETF protocols. It is clear that a specification "needs to be reasonably implementation independent"[2] and can leave some decisions to software engineers; however, we believe that any specification

[2] `http://www.ietf.org/iesg/statement/pseudocode-guidelines.html`

should be unambiguous and clear enough for guaranteeing the same behavior if it is given to different developers [7].

2.5 Optimized Link State Routing (OLSR) Protocol

Optimized link state routing (OLSR) is a proactive routing protocol for MANETs. [3] This protocol has the benefit of having the routes available when needed because of its proactive nature. The underlying mechanism of this protocol is the periodic exchange of messages to find routes. Since OLSR reduces the control packets size and minimizes flooding of this control traffic, it is known as optimization of a pure link state protocol. Moreover, this protocol does not generate extra control traffic in response to link breakage or failure. Every node stores the routes to all destinations in the network. As a consequence, it is applicable where a large subset of nodes are communicating with each other or nodes are changing with time. The protocol is specifically appropriate for large and dense networks as more optimization is achieved. OLSR works in a completely distributed manner without depending on any central entity. A reliable transmission is not needed for its control messages, since sending these messages occurs periodically [18].

The protocol minimizes flooding of control messages in the network selecting Multipoint Relays (MPRs), one-hop neighbors that retransmit received messages from the sender node. Every node, lets say node n, in the network attempts to choose a set of nodes among its one-hop neighbors. The selected nodes in the set can retransmit the received packet from node n, if they have a link toward two-hop neighbors of node n [18].

There are two types of control messages for this protocol, HELLO and Topology Control (TC) messages used for detecting neighbors and building topological information, respectively.

2.5.1 HELLO Message

Every node detects its direct neighbor nodes by sending a HELLO message every 2 seconds. A HELLO message traverses only one wireless link or a single hop and it is not forwarded. Every HELLO message contains: message type, the originator address, message valid time and one-hop neighbors of the originator [13]. Since every HELLO message contains one-hop neighbors of the originator, the receivers of HELLO messages can learn about two-hop neighbors.

[3] In WMN backbone, two nodes can communicate directly over one-hop wireless link, when they are in transmission range; otherwise, they have to communicate over multi-hop wireless link using other nodes as intermediate nodes. In this case, WMNs backbone is similar to the MANET. This bears the feature of multi-hop wireless communication for MANET; however in most cases, nodes in WMN can be stationary. As a consequence, MANET routing protocols such as OLSR is applicable to WMN backbone [17].

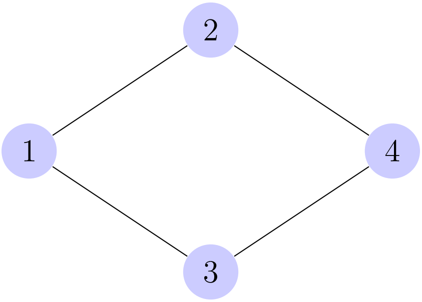

Fig. 2.8. A sample network of 4 nodes.

After receiving a HELLO message by a node, the information in the routing table[4] for one-hop and two-hop neighbors is updated [13]. The information accumulated in the routing table is updated according to the information in the HELLO message. Fig. 2.8 shows a sample network consists of 4 nodes. When nodes start broadcasting HELLO messages, they do no have any information about their one-hop neighbors. Therefore, they only send their own address as the message originator and the valid time of the message together with the type of message in their broadcasted HELLO messages. For instance, node 2 receives the HELLO message of node 1 and 4. The HELLO message of node 1 has the following information:

- HELLO is the type of the message.
- 1 is the originator of the message.
- 6 seconds is the validity time of the message. It represents that the broadcasted message is valid for 6 seconds.

This node does not have any information about its one-hop neighbors yet. Node 4 HELLO message contains the information as following:

- HELLO is the type of the message.
- 4 is the message originator.
- 6 seconds is the message valid time.

nodes	hops	next node	last sequence number	one hop neighbors
1	1	1		
2				
3				
4	1	4		

Table 2.1. Node 2 updated routing table after receiving first HELLO from nodes 1 and 4.

[4] Every routing table contains the address of nodes in the network, number of hops from every node, next node along the path to other nodes, last sequence number received from other nodes and one-hop neighbors of nodes.

Upon receiving these messages, node 2 updates its routing table as depicted in table 2.1. It sets the number of hops to 1, since it has received the HELLO message from its one-hop neighbors, and updates `next node` by adding the address of message originator to the table. Then, node 2, broadcasts its HELLO messages to its immediate neighbors. Nodes 1 and 4 receive the messages and update their tables. The HELLO message sent by node 2 contains this information:

- HELLO is the type of the message.
- 2 is the originator address.
- 6 seconds is the validity time of the message.
- 1 and 4 are one-hop neighbors of node 2.

Table 2.2 represents routing table of node 1 after being updated.

nodes	hops	next node	last sequence number	one hop neighbors
1				
2	1	2		1,4
3				
4	2	2		

Table 2.2. Node `1` updated routing table after receiving HELLO from nodes 2.

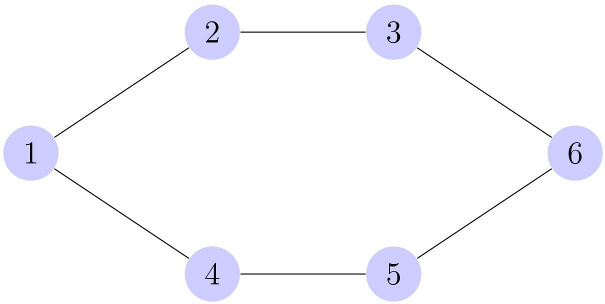

Fig. 2.9. A sample network of 6 nodes.

Table 2.2 indicates the number of hops to nodes 2 and 4 which are 1 and 2, respectively. Moreover, it shows if node 1 wants to deliver a packet to these two nodes, the next node in the path is node 2. The `one hop neighbors` in the routing table represents the one-hop neighbors of every node. In this example one-hop neighbors of node 2 are nodes 1 and 4. The `last sequence number`[5] in the routing table remains empty, since HELLO messages do not have sequence numbers. This column is updated

[5] Sequence numbers are used for the purpose of dropping old messages. In other words, if one node receives a message from another node, first it checks whether or not the message is new. In case, the message has been received before via this node, it will be dropped.

while receiving TC messages. TC message is another type of control messages used for updating topological information for more than one hop, which will be explained later in this chapter.

Fig. 2.9 depicts a lager network with 6 nodes. In this network, we explain process of sending HELLO messages by nodes 5 and 3, and also updating routing table of node 6 for these nodes. We should mention here that nodes 3 and 5 have already updated their routing tables for their immediate neighbors in the same way described for node 1 in 4 nodes network. Table 2.3 demonstrates the information of node 6 about its one-hop and two-hop neighbors after receiving HELLO messages from nodes 3 and 5. The distance of node 6 to nodes 3 and 5 is 1, since they are one-hop neighbors of this node. Two-hop neighbors of node 6 are nodes 2 and 4. The next nodes on the path to node 2 and 4 are nodes 3 and 5, respectively.

nodes	hops	next node	last sequence number	one hop neighbors
1				
2	2	3		
3	1	3		2,6
4	2	5		
5	1	5		4,6
6				

Table 2.3. Node 6 updated routing table after receiving HELLO from nodes 3 and 5.

The process of updating tables repeats also for other nodes in the network.

2.5.2 Topology Control (TC) Message

Nodes broadcasts particular control messages known as Topology Control (TC) messages every 5 seconds to create the *intra-forwarding* database required for routing packets. A TC is used to discover routes longer than a single hop and is forwarded in the entire network. It is employed for producing and changing topological information. These messages provide the required information for nodes to build routing tables [18]. Every TC message contains the following elements: message type, message originator, message sequence number, message Time To Live (TTL) which is equal to number of nodes in the network represents how many hops the TC message can be retransmitted, hops or distance from the originator, validity time of the message, advertised neighbors main address (one-hop neighbors of the originator) [13]. As long as all nodes in the network have the same purpose of finding routes to the destination, they update their routing tables to calculate and find the best paths to other nodes. This is achieved by broadcasting, receiving, processing and forwarding TC messages. As we mentioned, Fig. 2.9 demonstrates a network of 6 nodes. In this figure, we assume node 1 broadcasts its TC to its neighbors. The receiver nodes will update their tables and then forward the message to the next nodes. Finally, node 6 receives the message and updates its table for node 1. The TC message transmitted by node 1 contains:

- TC is the message type.
- 1 is the message originator.
- 1 is the message sequence number.
- 6 is the TTL of TC message.
- 0 represents distance from the originator.
- 15 seconds is validity time of TC.
- $2, 4$ are one-hop neighbors of node 1.

Table 2.4 shows the routing table of node 6. While receiving TC from node 1, as the sender, to node 6, as the receiver, the routing table for node 6 is updated. The `hops` from node 6 to node 1 is 3, the next node to reach node 1 is node 3[6], the last sequence number received from node 1 is 1 and finally, one-hop neighbors of node 1 are 2 and 4, as depicted in this figure. We should mention here that node 6 has already updated its table for nodes 2, 3, 4 and 5 when receiving HELLO messages sent by nodes 3 and 5, see table 2.3.

nodes	hops	next node	last sequence number	one hop neighbors
1	3	3	1	2,4
2	2	3		
3	1	3		2,6
4	2	5		
5	1	5		4,6
6				

Table 2.4. Node `6` updated routing table after receiving TC from node 1.

2.5.3 Core Functionality

The behavior of a node in the network running the OLSR protocol for routing can be specified as the core functionality. Moreover, a universal specification of the OLSR messages and their broadcasting through the network together with neighbor detection and route calculation are included [13]. Core functionally determines if a protocol is able to provide routing in a network. The following components are making up the core:

- Message formats
- Neighbor detection
- Topology control message diffusion

[6] Since node 6 has received the TC message of node 1 via node 3, it stores node 3 as the next node to node 1.

- Route calculation

Nodes in the network start working by broadcasting HELLO messages to their neighbors to detect direct one-hop neighbors and also two-hop neighbors. Then, nodes will try to create and send their TC message in the entire network. These messages then allow nodes to update their routing tables for different nodes in the network. To construct the routing table of a node, a shortest path algorithm is used [13]. It means that this routing protocol selects shorter paths by applying *Dijkstra's algorithm* [19]. Every routing table consists of destination addresses, next nodes along the path to the destination, distance from the node to the destination, etc [13].

Upon receiving a TC message, a node must perform some tasks for each message. Following shows those tasks according to [13]:

1. If the message was sent by the receiving node, the message MUST silently be dropped.

2. If the receiver node is not the message originator and the sequence number existing in the message is smaller than the one recorded in the receiver node's routing table, the further processing of this TC message MUST NOT be performed and the message MUST be silently discarded (message is out of order) [13].

3. If the receiver node is not the message originator and the sequence number existing in the message is equal to the one recorded in the routing table, the message has been completely processed and MUST not be processed again [13]. By considering the shortest paths algorithm, if hops of the message is smaller than the hops in the routing table, the message can be processed or even forwarded.

4. If the receiver node is not the message originator and the sequence number existing in the message is greater than the one recorded in the routing table, the new sequence number of the message is replaced with the old one in the routing table [13].

5. If the receiver node is not the message originator and the sequence number existing in the message is equal to the one recorded in the routing table, and the message has not been forwarded before, the message will be considered for forwarding [13].

6. If the message is considered for forwarding, time to live and hops of that message will be checked. If time to live of the message is greater than one and hops of the message is smaller than the number of nodes in the network, and the message has not been forwarded before, it MUST be retransmitted. After retransmitting, the message is marked as retransmitted, its time to live is reduced by one and hops of the message is increased by one [13].

7. If there is a change in one-hop neighbors of a node, the node's sequence number must be incremented by one [13].

After broadcasting HELLO and TC messages regularly and if the network does not change, all nodes have the topological information of the entire network, such as distance from different nodes, one-hop neighbors of every node, next node to the destination, etc. In this situation, routes to all destinations are already established. In this study, we formalize both two kind of messages and the process of sending, receiving, processing and forwarding. In general, we make a formal model of core functionality of the OLSR.

Chapter 3

Formal Methods

Formal methods, mathematical-based techniques useful for specification, development and verification of systems, target at enhancing the rigor of the design and development of systems [20]. In these approaches, a language with mathematically defined syntax and semantics is applied to create a formal specification [21]. Timed automata is a theory applied for this purpose. Then the system properties like functional behavior, performance characteristics or timing behavior are verified based on that formal specification [21].

Formal verification should cover all possible behaviors of the system—not only the ones which appear most likely in real life. By this, even unpredictable events can be covered. In other words, formal verification is used to analyze one system for its required properties. The main noticeable benefit of formal techniques is making clear and apparent assumptions that leads to producing correct outputs [22]. An important field is model checking where a given property is checked against a given model.

In this chapter, we first give an overview of the theory of timed automata and in particular Uppaal timed automata which is used for creating a formal model of OLSR. Then we explain model checking [23] technique which is applied to decide automatically whether a desired property is satisfied or not. At last in this chapter, we describe the Uppaal model checker in detail.

3.1 Timed Automata

Timed automata is a theory to model, analyze and verify the timing behavior of real-time systems and networks. A timed automaton is a finite automaton, a graph consisting of a finite set of locations and transitions, together with a finite set of clocks with real values. It can be considered as a model which is an abstraction from a timed system. The logical clocks in the system are initialized with zero at start, and all clocks increase their value implicitly with the same rate as time progresses while running the timed automata [24]. Guards on transitions are used to restrict the automatons behavior. For instance, one transition can be taken if the clock value or the constraints on that transition are satisfied. As soon as one transition is taken, clocks may be reset to zero. The formal definition of a timed automaton is a tuple with the following elements ($\mathbf{L}$, $\mathbf{l_0}$, $\mathbf{C}$, $\mathbf{A}$, $\mathbf{E}$, $\mathbf{I}$) [25].

- $\mathbf{L}$ is a finite set of states or locations.
- $\mathbf{l_0}$ is called initial state which is one element of $\mathbf{L}$.
- $\mathbf{C}$ is a finite set of clocks.
- $\mathbf{A}$ is a set of actions, co-actions and internal actions.
- $\mathbf{E} \subseteq \mathbf{L}\times\mathbf{A}\times\mathbf{B}(\mathbf{C})^1\times\mathbf{2^C}\times\mathbf{L}$ is a set of edges between locations with actions, guards and a set of clocks (to be reset).
- $\mathbf{I} : \mathbf{L} \to \mathbf{B}(\mathbf{C})$ are invariants on locations.

In general, going from state $\mathbf{l_1}$ to state $\mathbf{l_2}$ is possible by taking a transition from $\mathbf{E}$ considering the guards and actions on the same transition.

The semantics of timed automata is defined as follows:

"A clock valuation is a function $u : C \to \mathbb{R}_{\geq 0}$ from the set of clocks to the non-negative reals. Let $\mathbb{R}^C$ be the set of all clock valuations. Let $u_0(x) = 0$ for all $x \in C$. The notation is abused by considering invariants and guards as sets of clock valuations, writing $u \in I(l)$ to mean that u satisfies $I(l)$" [25, p.3].

If ($\mathbf{L}$, $\mathbf{l_0}$, $\mathbf{C}$, $\mathbf{A}$, $\mathbf{E}$, $\mathbf{I}$) is a timed automaton, the semantics is described as:

"a labelled transition system $\langle S, s_0,\to\rangle$, where $S \subseteq L\times\mathbb{R}^C$ is the set of states, $s_0= (l_0, u_0)$ is the initial state, and $\to \subseteq S\times (\mathbb{R}_{\geq 0}\cup A)\times S$ is the transition relation such that:

- $(l,u) \xrightarrow{d} (l,\ u + d)\ if\ \forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(l), and$

- $(l,u) \xrightarrow{a} (l',\ u')\ if\ there\ exists\ e = (l, a, g, r, l') \in E\ so\ that\ u \in g,$

$u' = [r \mapsto 0]u,\ and\ u' \in I(l)$, where

for $d \in \mathbb{R}_{\geq 0}$, $u + d$ maps each clock x in C to the value $u(x) + d$, and $[r \mapsto 0]u$ denotes the clock valuation which maps each clock in r to 0 and agrees with u over $C \setminus r$ " [25, p.3].

Uppaal timed automata, the tool used in this study, has some additional features which are described in the following section.

3.2 Uppaal Automata

The Uppaal modeling language expands timed automata adding some additional features, see [25] . Uppaal automata provide "data structure with several types, variables ranging over these types, operators and predicates. Common Boolean and arithmetic expressions are used to denote data values and statements about them" [12, p.4]. "The state of the system is determined, in part, by the values of data variables that can be either shared between automata, or local" [12, p.4]. Every automaton is a graph with locations and edges between these locations together with guards and clock constraints. Each edge has a guard, a synchronization label, and an update,

[1] $\mathbf{B}(\mathbf{C})$ is a finite set of guards on transitions.

optionally. Guards on transitions are used to restrict the automaton behavior. Synchronization happens via channels; for every channel a there is one label $a!$ to identify a sender, and $a?$ to represent a receiver. Transitions with no labels are internal transitions and all other transitions use one of two following types of synchronization [12].

In *binary handshake* synchronization, one automaton which has an edge with a label ! synchronizes with another automaton with the edge having a label ?. These two transitions synchronize only when both guards evaluate to true in the current state. After taking the transitions, both locations will change, and the updates on transitions will be applied to the state variables; first the updates will be done on the !*edge*, then the updates occur on the ?*edge*. In case of having more than one possible pair, the transition will be selected non-deterministically [12].

In *broadcast* synchroniation, one automaton with a !*edge* synchronies with several other automata that all have an edge with a relevant ?*edge*. The initiating automaton is able to change its location, and apply its update, if and only if the guard on its edge is satisfied. It does not need a second automaton to synchronize with. Matching ?*edge* automata must synchronize if their guard is true, currently. They will change their location and do the updating of the state. At first, the automaton with the !*edge* will update the state, then the other automata will follow in some lexicographic order. When more than one automaton can initiate a transition on an !*edge*, the process of choosing will occur non-deterministically [12].

3.3 Model Checking

The verification technology of model checking "provides an algorithmic means of determining whether an abstract model—representing, for example, a hardware or software design—satisfies a formal specification expressed as a temporal logic formula [26, p.1]. If the property is not satisfied, this method identifies a counterexample execution which indicates the source of the problem [26]. As soon as problems are identified (by means of counterexamples), those problems can be analyzed and hopefully solved.

Temporal logic is divided into two categories: Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) which is used by Uppaal. CTL uses $\mathbf{A}$ and $\mathbf{E}$ as path quantifiers, and $\mathbf{G}, \mathbf{F}, \mathbf{X}$, and $\mathbf{U}$ as temporal operators. Here, a path contains an infinite sequence of states which are connected using transitions. "The (state) formula $\mathbf{A}\phi$ is satisfied in a state if all paths starting in that state satisfy ϕ, while $\mathbf{E}\phi$ is satisfied if some path satisfies ϕ" [27, p.17]. The (path) formulas $\mathbf{G}\phi$ means ϕ holds globally in all states, $\mathbf{F}\phi$ represents ϕ holds eventually in some state, and $\mathbf{X}\phi$ indicates ϕ holds in the next state of a path. Finally, "the *until* $\phi\mathbf{U}\psi$ means that, until a state occurs along the path that satisfies ψ, property ϕ has to hold. In CTL, a temporal operator must always immediately be preceded by a path quantifier" [27, p.18].

Model checking techniques bear more advantages than just counterexample generation:

- Model checking is fully automatic. As soon as a model has been derived and a property has been stated, the analysis is done by a computer solely; so no human has to interfere. By this, model checking techniques are among the most efficient tools for analyzing systems. The most important and maybe difficult issue is to state the property for the system.

- The automatic analysis is usually fast and returns results quickly. This is in contrast to other formal methods such as interactive theorem proving. However, the price to be paid is (a) in the setting of WMN routing protocols only a finite number of topologies can be analyzed; it is hardly possible to verify properties for all possible topologies (this would be require some sophisticated new abstraction techniques and is far beyond this project); and (b) is limited to systems with a "small" state space. State space explosion is probably the most limiting factor of model checking. As soon as the model becomes too large, it is not possible to analyze and check all possible behaviors of a systems—even with today's fast machines. Due to the proactive nature of the OLSR and the high amount of flooding messages in the network, we can run our system on small topologies up to 4 nodes; here state space explosion should not be an issue.

- Checkers use temporal logics to express and formulate properties. Temporal logics are well known and it is common knowledge that temporal logics are most sensitive for real-time systems that require high level of accuracy and insurance. Moreover, since temporal logics are able to characterize (all) correctness properties we are interested in, Uppaal seems a perfect tool.

3.4 Uppaal

The tool that we use for analyzing OLSR is the model checker Uppaal. Uppaal is a tool for modeling, simulating and verifying real-time systems. It is developed jointly by Uppsala University and Aalborg University. This tool is designed for systems that can be modeled as networks of timed automata and it is used to verify such systems. Uppaal is a commercial tool (free for academics) which has been used in various case studies ranging from communication protocols to multimedia applications.[2]

Verification in Uppaal is usually performed in three steps. The first step is the creation of a model of the system; the second step is simulating the model and in the last stage, verifying properties of the model is done [25].

- The first step allows the user to represent the system as a set of extended timed automata with guards, described in 3.2. These networks of automata are the specification and description language of Uppaal.

- The second step is a "manual" check of the model using the simulator. System validation is done in this step. The simulator of Uppaal can be used to examine

[2] For an overview on Uppaal and the case studies we refer to the Uppaal webpage `http://www.Uppaal.org`

possible dynamic behaviors of the system and it provides an inexpensive means for fault detection [28]. The simulator is very useful to check that the system models act as expected and to obtain (new) insight of the system; it does not verify properties. For our analysis this step is an intermediate step to fix bugs of our model.

- The third and final step applies "real" model checking. Verification of the system is performed in this stage. It checks whether the system behaves as it is designed to behave. Uppaal's verifier uses CTL [29] to state system properties. The query language in Uppaal contains two formulas, path formulae and state formulae as defined in section 3.3.

Chapter 4

Related Work

Modeling and verifying protocols have been done for years already; however attempts to verify routing protocols for WMNs are still rather new and remain a challenging task. Traditionally routing protocols for WMNs are implemented and deployed in test-beds, simulations and in reality. By doing so, the protocols can be analyzed. However, the analysis is always limited to very few topologies [11]; moreover if a shortcoming is found it is often unclear whether the limitation is a consequence of the routing protocol chosen, or of the underlying link layer (the reason is that often both layers are implemented at the same time and that no clear separation is established).

In the project we use the model checker Uppaal to analyze OLSR. By creating a model of OLSR, we can abstract from the underlying link layer; hence a shortcoming found is definitely a problem of the routing protocol. Model checking techniques have been applied to analyze protocols since decades, but there are only a few papers that use these techniques in the context of mobile ad-hoc networks, e.g., [30]. In the area of WMNs, Uppaal has been used to model and analyze the routing protocols AODV and DYMO, see [11], [31], [12].

In this chapter, we point at some related work about the OLSR protocol. In particular we concentrate on work on modeling and verifying this protocol. We will also mention formal methods that are related and are used to analyze similar protocols, such as AODV. However, to the best of our knowledge, the project will be the first, aiming at a formal model of OLSR core functionality considering time variables.

4.1 OLSR Modeling and Verification

Clausen et al. [13] specify the OLSR routing protocol, which is used in mobile ad-hoc networks. This paper is the official description and is the document currently standardized by the IETF.

Jacquet et al. [18] also provide a higher-level description of OLSR. This study gives an overview of OLSR; in particular it describes the advantages of this protocol (when compared to the others). First advantage of OLSR is having the information about all nodes in the network. In other words, information about possible routes or paths to the destination is available due to flooded information in the network. Hence, data packets that need to be sent to a destination can be transmitted straight away. One of the other advantages of this protocol is its suitability for large and dense networks.The authors also mention that this protocol is considered as optimization of a pure link state protocol for two reasons: reduction in control packets size, minimization of control messages traffic using MPRs.

Steele and Andel [32] provide another study of the OLSR and describe a framework for modeling and verifying of this protocol. The authors use formal methods to verify properties of the OLSR protocol. They designed a model of OLSR in which LTL is used to analyze the correct function of this protocol. Spin, a useful tool that provides an environment for modeling and verifying distributed systems, used in their study. Spin is a model checker which is not based on timed automata. Their approach comprises networks of 4 nodes that need to be verified for loop-freedom, accurate neighbor discovery, and relay selection. They could prove that their model is able to satisfy these properties and also it can verify OLSR in case of including a Byzantine failed node, a node which disrupts a routing protocol. In addition, this simple Byzantine failure represents the ability of their OLSR model to discover protocol property violations. One of the existing issues of their model is about the accuracy of created routes by OLSR. The considered properties of their OLSR model do not verify this and it needs adding data exchange to the model. Moreover, they did not consider timing parameter in their model due to disability of Spin to use time. They have only examined an instant in time. Since proactive protocols highly depend on on-time broadcasting of control messages, this can be considered as an issue in their model. In addition, they have not considered the most time consuming activity, broadcasting control messages, which may cause appearing deadlock in their model. Also, it is not possible to state properties associated with timing by spin which can easily be done by Uppaal.

4.2 Modeling and Verification of AODV Using Uppaal

Since we use the model checker Uppaal, we discuss how Uppaal has been used to model, analyze and verify other routing protocols in the area of WMNs.

Fehnker et al. [12] describe a formal and rigorous model of AODV routing protocol by Uppaal which is derived from a precise process-algebraic model that reflects a common and unambiguous interpretation of the RFC [33]. They model each node in the network as an automaton that has a routing table and message buffer. There are 4 type of messages, such as PKT, RREQ, RREP and RERR in their model.

Their experiment contains four scenarios with two data packets and 5 nodes in the network. All these four scenarios have been implemented using a simple automaton called tester. Then, three different properties have been investigated. The first property is that when all routing messages are processed, a route has been found from the originator to the destination. The next property checks once all messages have been processed, no sub-optimal route has been found (number of hops is greater than the shortest path). The third property represents that “no sub-optimal routes will be found at all” [12, p.9].

In the next part of this study, the experiments have been done to quantify the number of topologies that are influenced by two problems of AODV, finding non-optimal routes and disability in finding any routes, and the modifications were discussed. This section consists of three proposed variants of AODV:

- forwarding all route replies
- replying to improving requests
- recovering from failed replies

According to the RFC specification of AODV, RREP messages are discarded by intermediates nodes. These intermediate nodes send the RREP messages when they are not the originator node. Discarding of the RREP messages can happen when a node has to disseminate several route requests for a specific destination. One possible solution is to forward each reply received by a node. However this increments the number of control messages, it decreases the need for sending route requests. The experiment shows that the problem is addressed by this modification. Also, counterexamples by Uppaal show that AODV is able to reply to the first route request and it will ignore all other subsequent requests that have the same request ID, even the ones arrived via shorter routes. The possible modification in here is replying to the first request together with the subsequent request with an improved hop count. Recovering from failed replies is used as the solution for the main reason of failed route discovery, marking a request as a replied one even if it has been detected by the node that the reply failed because of breaking link. The modification is done by not marking as the seen request if the reply fails, and replying to other route discovery requests. The main reason for route reply messages to get lost is breaking intermediate links on the way back to the destination. The possible solution to overcome this problem is to store a set of routes or to perform different error responses that needs a huge changes in AODV characteristics.

Fehnker et al. [11] have used Uppaal model checker to model the behavior of the AODV routing protocol and verify some properties of this protocol according to its specifications. They could prove some possible problems and unusual behavior of this protocol. They discovered in case of broadcasting, some properties of AODV will not hold. For instance, the establishing routes property will not be always satisfied. In their study, there are four types of messages, PKT, RREQ, RREP, RERR. They also assumed the dynamic topology that let ruining or establishing connections. They designed their model in such a manner that if one node receives a packet, it will queue it and then the node will send a RREQ message to its neighbors. Next, if receivers have the valid and fresh connection to the destination they will respond with the RREP message. RREP message is an unicast message that goes back to the originator of RREQ message. In case of failing the RREP message in the middle of way, the route will not be established.

They assumed to have 3 nodes; `s`, `a`, `d`, which are connected to each other in the network. Node `s` and `a` aim at delivering packets to the destination, node `d`. First node, let say node s, sends a RREQ message to its neighbor, let say node a and also node a is targeting at sending a RREQ message to node s and d. Node d will respond to the RREQ message from a by RREP message and since node a has found the way to destination which is node d, it will deliver its packet. But, node s has not received the RREP message from node a, so it will not establish a route to node a and cannot deliver its packet. By applying CTL, it has been proved that AODV does not always establish routes. In addition, another property has been satisfied in the study, producing non-optimal routes. They presumed that they have 5 nodes and first node can send its packet to the destination node by crossing 2 hops. But, the packet will be delivered to the destination after passing 3 hops. It can be concluded that the route is not an optimal route. This scenario has been checked in this paper and finally it has been demonstrated that AODV sometimes creates non-optimal routes to the destination.

Although the two protocols, AODV and OLSR, behave different; it is possible to adapt the modeling techniques used for AODV. We use the same modeling techniques and the same experiments as for AODV.

4.3 Differences Between OLSR and AODV

Huhtonen [34] has compared AODV and OLSR. This paper gives a general knowledge about mobile wireless networks and the explanation about different routing protocols are used in such networks that also has been mentioned in [18]. The mechanism and how AODV and OLSR work in mobile wireless networks and the difference between these two were the goal of the author. The paper compares these two protocols in terms of performance and scalability, resource usage and security.

One of the benefits of OLSR was pointed in their study is having the routing information about all participated nodes in the network. Another benefit is that reliability of links is not a requirement for the control messages due to periodic sending messages; hence OLSR can react quickly on topology changes. But AODV outperforms OLSR in case of facing resource critical environments and security considerations.

We are focusing on modeling OLSR with the same tool used for modeling AODV. By this, we also pave the way to another comparison between AODV and OLSR in our future work.

Chapter 5

Modeling OLSR in Uppaal

In section 2.5, we have sketched the OLSR protocol and discussed the core functionality of this protocol. In this chapter, we describe our Uppaal model in detail; however, we have to make some assumptions to only focus on verifying the main functionality of the protocol and retaining our specification manageable. In this chapter, we first describe the system boundaries and assumptions and then talk about the principles of our model. At last in this chapter, we discuss about different network topologies which the OLSR can be implemented on.

5.1 System Boundaries and Assumptions

Every model is an abstraction of a real system. Most often a model assumes perfect scenarios. An example is the incrementation of natural numbers; in the model this is always possible since there are infinitely many numbers. In any implementation, however, at some point a number is hit, which cannot be incremented (storing numbers consumes allocated memory and at some point all the memory is used). By this, when creating the model of OLSR, we have to make reasonable assumptions on the system. Our OLSR formalization attempts to accurately model core functionality of the protocol as defined in the IETF RFC 3626 specification [13].

One abstraction is concerned with OLSR itself. This routing protocol is a complex system, with a specification of about 100 pages of text. However, all details are not relevant to the main functionality. Therefore, we concentrate on the core functionality that is always required for the protocol to perform. We have also abstracted from all optional features of OLSR. This retains our specification manageable.

The restriction to the core functionality is to avoid the biggest limitation of model checking: state space explosion. State space explosion will restrict the details of the model, as well as the size of topology. Experience shows that model checking of WMNs is usually limited to topologies up to 10 nodes [31]. In reality, the number of nodes can easily reach several hundreds. At first glance, it might seem that analyzing small networks is not sufficient and that problems of networks only occur in large ones. Many years of case studies and industrial experience, however, shows that is not the case. It appears that fundamental flaws in system such as WMNs can be found in small scenarios [11].

Moreover, all links are assumed to be bi-directional. The MANET community commonly makes this assumption. The model of OLSR can be simplified by ignoring unidirectional links, however it is left as our future work to develop our OLSR model to non-bidirectional environments.

Due to the fact that the main goal of this study is showing the feasibility of constructing and verifying a correct and concrete model of OLSR by model checking technique, for all networks, only static ones are considered where link failure will not occur. Moreover, as long as nodes in WMNs can be stationary, having static network is considered as a correct assumption. However, modeling mobility is left as our future work.

5.2 OLSR model

OLSR seeks to store a constantly updated topology information and the whole network has to be known to all nodes. In our model of OLSR, we design a routing table for every node[1] to maintain information about all destinations. Upon receiving a message by a node, its routing table is updated according to the information in the received message. Routing tables provide all the information required for route establishment and packet delivery. Every routing table existing in each node has:

- All the addresses of nodes in the network.
- Number of hops from that node to the destination nodes.
- Next nodes along the path to the destination nodes.
- Last sequence number of destinations that their message received by the node. The freshness of TC messages received by this node can be checked by this entry.
- There are flags[2] in the routing table for every destination which indicates if their received TC message has been forwarded before. After retransmitting TC messages, these values are set to 1.
- One-hop neighbors of destination nodes that are used to update two-hop neighbors of those destinations while receiving HELLO messages and also they are used to give the information about one-hop neighbors of all destinations while receiving TC messages.

We use Uppaal to model OLSR as a parallel combination between node processes. Every process is also a parallel composition of three timed automata shown in the following:

[1] All nodes in the network are MPR nodes that are able to transmit HELLO and TC messages. These nodes can forward TC messages as well.

[2] As described in section 2.5.3, if a TC is retransmitted before by a node, it must be marked as retransmitted. We model this by defining a flag for every destination.

1. The Controller is used to model on-time broadcasting of control messages.

2. The OLSR which is the behavior of the main nodes.

3. The Queue that has been chosen to store incoming messages from other nodes. In other words, it denotes the input buffer of a node. The received messages are buffered and in turn are sent to the OLSR automaton for processing.

In this chapter, we explain these processes in details. All of these automata have their own data structures. For instance, the OLSR automaton has a routing table rt as described earlier in this chapter, which is an array of entries. Every entry is shown by the data type

```
typedef struct {
IP dip; // Destination address.
int hops; // Number of hops to the destination.
IP nhopip; // Next hop address along the path to the destination.
SQN dsn; // Last sequence number of TC originator.
bool flag; // If TC msg was forwarded before.
bool onehop[N]; // one-hop neighbors of TC or HELLO originator.
} rtentry;
```

IP denotes a data type for all addresses and SQN represents one for sequence numbers. OLSR uses sequence numbers to check whether received messages are new or they have been processed before. In our model, integers are used to define these types[3].

Every message is a struct with elements:

```
typedef struct {
MSGTYPE msgtype;
IP oip;
IP dip;
bool onehop[N];
int ttltc;
VTIME_H vtime_h;
VTIME_TC vtime_tc;
int hops;
IP sip;
SQN osn;
} MSG;
```

The description of each element in this structure is as follows:

- msgtype shows type of messages flooded in the network and can have values PACKET, HELLO, or TC.

- oip is the originator address and represents the originator of the message.

- dip is the destination address and indicates the destination of the message. We use this elements only in the PACKET.

[3] In the RFC 3626, the address of each node is IPv4 address and sequence numbers are integers.

- onehop is a boolean array of size N[4] exists in HELLO and TC messages which shows one-hop neighbors of the originator.

- ttltc is an integer equals to the number of nodes in the network which represents how many hops the message can be transferred. Every time a message is forwarded, this value is decremented by 1. We use this element for TC messages.

- vtime_h is HELLO message valid time[5] equals to 6 seconds.

- vtime_tc is TC message valid time equals to 15 seconds.

- hops is an integer indicates the distance from the originator of the message to the receiver node. Every time a message is forwarded, this value is incremented by 1. We need hops only in TC messages.

- sip is the sender address of a message and is used to determine nhopip in routing tables.

- osn is the message sequence number shows the freshness of the message. The originator node assigns this identification number to each TC message.[6]

While creating HELLO, TC and PACKET this structure is used. Communication between two nodes is possible only if they are in transmission range of each other which is modeled by the predicate isconnected[7]. If isconnected evaluates to true, then nodes can communicate on channels which are named according to the message type being delivered (hello, tc). We define several channels in our model that allow different automata to communicate with each other that are described in detail.

- Channels hello and tc are used to model broadcast, we have one broadcast channel for each node. For instance, hello[ip] is used for broadcasting HELLO messages of node ip[8] and tc[ip] is applied to broadcast node ip's TC messages.

- Channels CreateHello and CreateTc are used for on-time broadcasting of control messages. They make it possible for the Controller to synchronize with the OLSR at a specific time. These two channels are broadcast channels.

[4] N is equal to the number of nodes + 1. We consider "+1" only for indexing.

[5] Every step (e.g., processing a message) inside a computer takes time; but this is so small compared to message sending. Hence, we assume while processing messages, time does not progress to exceed validity time. We model this by applying committed locations in Uppaal in which no delay occurs. We embed validity time in HELLO and TC messages to have a complete model.

[6] Every OLSR automaton has a local variable sn with type SQN. This value is incremented by 1 via the OLSR every time there is a change in one-hop neighbors of a node and is embeded in TC messages as osn.

[7]
```
bool isconnected(IP i, IP j){
if (topology[i][j]==1) {
return 1;}
return 0;}
```
Here, topology is a constant integer two-dimensional array.

[8] Address of every node is defined as ip. The type of ip is IP.

- Channels pkt and packet are used to model packet injection and packet transfer from source node to the destination node, generated by the user layer. They provide the situation for injecting and transferring a packet into the system and they are applied for our experiments. Urgent[9]unicast channel packet is used for synchronizing the Tester and the Queue when a packet must be injected, and unicast channel pkt is applied to transfer the injected packet hop by hop from source node to the destination node using the next hop information in the routing table.

- Channels imsg and tau have been used for internal activities. Urgent unicast channel imsg is applicable to transfer messages from the Queue to the OLSR and the urgent broadcast channel tau used for internal transitions allow optimizations and reduce state space explosion problem.

There is also one global clock called clk which shows the global timing of the whole system. The process of broadcasting and receiving HELLO and TC messages are discussed in the following sections. In this study, we investigate two different scenarios:

1. Every node has a different starting time in the interval [0, random_start).[10]

2. All nodes start working at exactly the same time.[11]

5.2.1 Different Starting Time

As we mentioned in section 2.5, OLSR is a proactive protocol which is based on sending periodic control messages. In the network, every node broadcasts its two types of messages, HELLO and TC, every 2 and 5 seconds [13],[12] respectively. Therefore, we define an automaton called Controller to manage on-time broadcasting of messages by the periodic sending of HELLO and TC at certain times. The automaton is responsible to synchronize with the OLSR at a particular time. By considering the fact nodes start at a random timing between [0, random_start), all nodes need to have their own Controller with its local clock t to estimate the broadcasting time for the OLSR and synchronize with it.

5.2.1.1 Broadcasting HELLO Messages

In order to model times when nodes start broadcasting HELLO messages, we define a time_between_hello=2000 constant and a random_start variable which is varied between [0, 10). The random_start provides means for modeling a realistic specification where nodes can start broadcasting at different times.

[9] Urgent channel is a channel where no delay must occur and the synchronization must happen immediately.

[10] In this scenario, every OLSR automaton has its own Controller and Queue.

[11] In this scenario, every OLSR has its own Queue, but there is only one Controller for the whole system.

[12] In our model, we convert all time values to milliseconds to have the same unit for all times in the system.

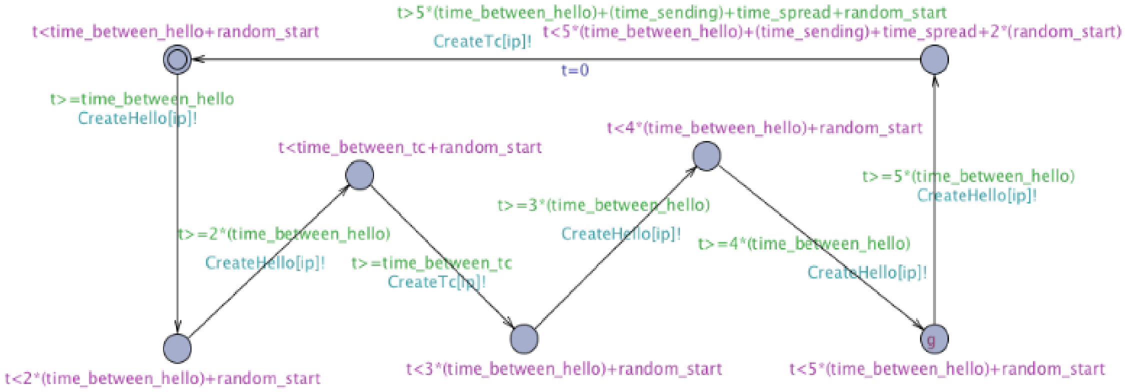
t<time_between_hello+random_start
t>5*(time_between_hello)+(time_sending)+time_spread+random_start
CreateTc[ip]!
t<5*(time_between_hello)+(time_sending)+time_spread+2*(random_start)
t=0
t>=time_between_hello
CreateHello[ip]!
t<time_between_tc+random_start
t<4*(time_between_hello)+random_start
t>=5*(time_between_hello)
CreateHello[ip]!
t>=3*(time_between_hello)
CreateHello[ip]!
t>=2*(time_between_hello)
CreateHello[ip]!
t>=time_between_tc
CreateTc[ip]!
t>=4*(time_between_hello)
CreateHello[ip]!
g
t<2*(time_between_hello)+random_start
t<3*(time_between_hello)+random_start
t<5*(time_between_hello)+random_start

Fig. 5.1. Different starting time Controller.

As depicted in Fig. 5.1, the first initial state of the Controller has the invariant t<time_between_hello+random_start that shows whenever t is smaller than 2010 milliseconds, the automata stays in this state until the guard on the transition, i.e., t>=time_between_hello, is satisfied. In other words, the transition can be taken if t reaches the time between [2000,2010) milliseconds. At this time if OLSR automaton is idle[13], the controller then will synchronize with the OLSR on CreateHello channel. No update or action will happen in the Controller and the only update is on the OLSR. It resets its local clock t[14] and changes its boolean idle to 0 that shows it is busy at that moment. After synchronization, both automata will go to their next state and wait there until guards on transitions are satisfied. Fig. 5.2 shows the OLSR automaton.

The Controller goes to the next state with invariant t<=2*(time_between_hello) and OLSR moves to the state with invariant t<=time_sending+time_spread where time_sending is a constant equal to 40 and constant time_spread equals to 5. The communication between nodes is the most time consuming activity, which takes on average 40 milliseconds.[15] So, OLSR will send its messages in the interval [time_sending - time_spread, time_sending + time_spread] which is [35, 45] uniformly at random. As long as t reaches this interval, OLSR and the Queue, depicted in Fig. 5.3, will synchronize on the hello channel, transferring the relevant message data from the OLSR to the Queue of isconnected nodes. The OLSR tries to create HELLO messages by the createhello function[16] using its routing table and copies the result of this function into a global variable msgglobal and finally updates its state to idle. As mentioned in section 2.5.1, every HELLO message contains:

- msgtype which is HELLO.

[13] idle in a boolean value for every OLSR automaton that represents its status. If it is equal to 0, the OLSR is busy with processing or creating messages. After processing, this value is set to 1.

[14] Every OLSR automaton has its own local clock called t to control consumed time for sending and forwarding messages.

[15] It is the experience of networking that sending messages take approximately 40 milliseconds.

[16] All functions are available in the appendix, see A.

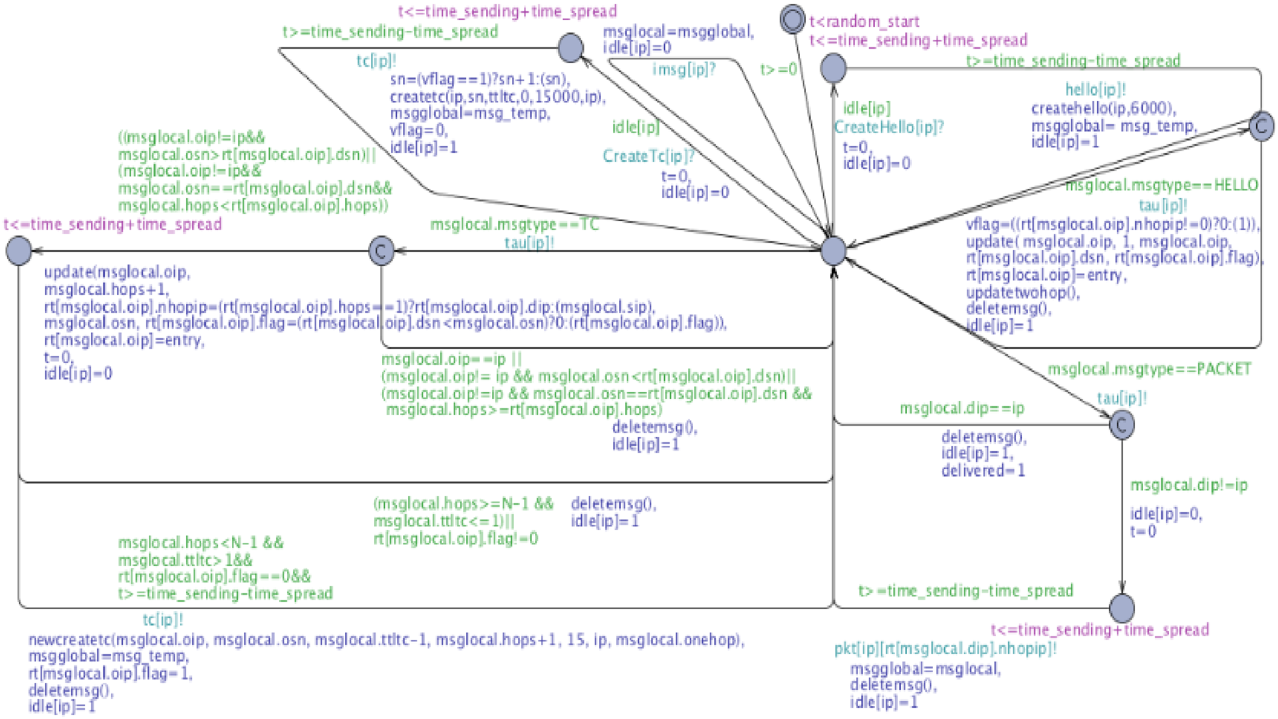
t<=time_sending+time_spread
t>=time_sending-time_spread
tc[ip]!
sn=(vflag==1)?sn+1:(sn),
createtc(ip,sn,ttltc,0,15000,ip),
msgglobal=msg_temp,
vflag=0,
idle[ip]=1
msglocal=msgglobal,
idle[ip]=0
imsg[ip]?
t>=0
t<random_start
t<=time_sending+time_spread
t>=time_sending-time_spread
idle[ip]
CreateTc[ip]?
t=0,
idle[ip]=0
idle[ip]
CreateHello[ip]?
t=0,
idle[ip]=0
hello[ip]!
createhello(ip,6000),
msgglobal= msg_temp,
idle[ip]=1
((msglocal.oip!=ip&&
msglocal.osn>rt[msglocal.oip].dsn)||
(msglocal.oip!=ip&&
msglocal.osn==rt[msglocal.oip].dsn&&
msglocal.hops<rt[msglocal.oip].hops))
t<=time_sending+time_spread
msglocal.msgtype==TC
tau[ip]!
msglocal.msgtype==HELLO
tau[ip]!
vflag=((rt[msglocal.oip].nhopip!=0)?0:(1)),
update(msglocal.oip, 1, msglocal.oip,
rt[msglocal.oip].dsn, rt[msglocal.oip].flag),
rt[msglocal.oip]=entry,
updatetwohop(),
deletemsg(),
idle[ip]=1
C
C
C
C
update(msglocal.oip,
msglocal.hops+1,
rt[msglocal.oip].nhopip=(rt[msglocal.oip].hops==1)?rt[msglocal.oip].dip:(msglocal.sip),
msglocal.osn, rt[msglocal.oip].flag=(rt[msglocal.oip].dsn<msglocal.osn)?0:(rt[msglocal.oip].flag)),
rt[msglocal.oip]=entry,
t=0,
idle[ip]=0
msglocal.oip==ip ||
(msglocal.oip!= ip && msglocal.osn<rt[msglocal.oip].dsn)||
(msglocal.oip!=ip && msglocal.osn==rt[msglocal.oip].dsn &&
msglocal.hops>=rt[msglocal.oip].hops)
deletemsg(),
idle[ip]=1
msglocal.msgtype==PACKET
tau[ip]!
msglocal.dip==ip
deletemsg(),
idle[ip]=1,
delivered=1
msglocal.dip!=ip
idle[ip]=0,
t=0
(msglocal.hops>=N-1 &&
msglocal.ttltc<=1)||
rt[msglocal.oip].flag!=0
deletemsg(),
idle[ip]=1
msglocal.hops<N-1 &&
msglocal.ttltc>1&&
rt[msglocal.oip].flag==0&&
t>=time_sending-time_spread
t>=time_sending-time_spread
tc[ip]!
newcreatetc(msglocal.oip, msglocal.osn, msglocal.ttltc-1, msglocal.hops+1, 15, ip, msglocal.onehop),
msgglobal=msg_temp,
rt[msglocal.oip].flag=1,
deletemsg(),
idle[ip]=1
t<=time_sending+time_spread
pkt[ip][rt[msglocal.dip].nhopip]!
msgglobal=msglocal,
deletemsg(),
idle[ip]=1

Fig. 5.2. `OLSR` automaton with its own `Controller`.

- `oip` that shows the message originator. The `OLSR` sets this value by assigning its `ip` into this element.
- `vtime_h` which is set to 6000 milliseconds and is the valid time of HELLO message.
- `onehop` is a boolean array of one-hop neighbors of the originator.

Incoming messages to a node are stored in its `Queue` using the `addmsg` function and are deleted from the `Queue` when are sent to the `OLSR` automaton by the `deletemsg` function. The `OLSR` status is busy/`not idle` when it broadcasts messages. It receives a new message from the `Queue` when it finishes handling a message, completely. In other words, when `OLSR` is `idle`, it is ready to process a new message waiting in the `Queue`. Whenever it is not busy with processing a message and there are some messages in the `Queue`, the `OLSR` and the `Queue` synchronize via the `imsg` channel in which the `Queue` and `OLSR` are sender and receiver, respectively. Then, the `Queue` copies the last message in the `Queue` into `msgglobal` and deletes that message by the `deletemsg` function from `Queue`. The `OLSR` copies the incoming message from its `Queue` into its local messages and changes its `idle` status to `not idle`. After receiving a HELLO message by a node,[17] the following processes occur as described

[17] The location called `C` in Fig. 5.2 represents the committed location. When the automaton is in this location, no delay is allowed and the automaton must take the next transition outgoing the committed location. It is used to guarantee that message processing does not take time, so then the

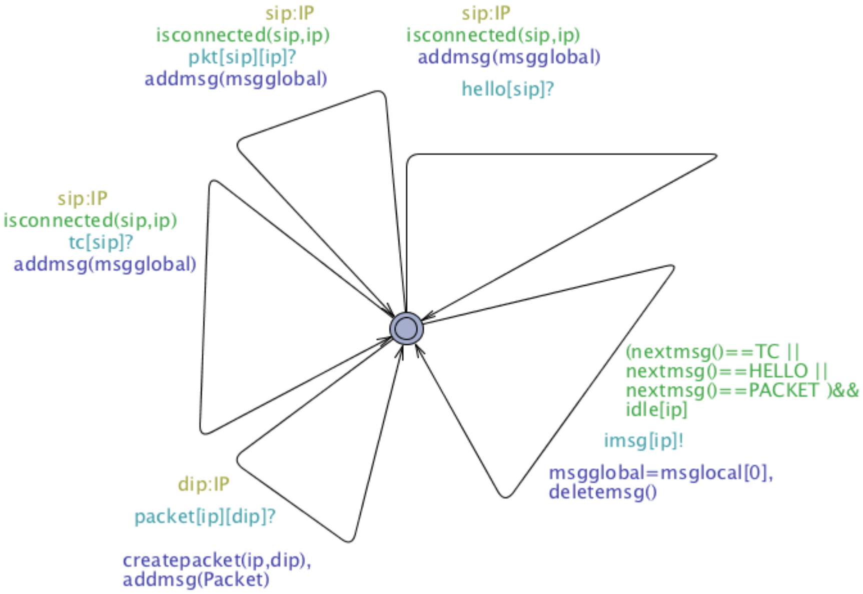

Fig. 5.3. `Queue` automaton.

in section 2.5.1:

- First, the `OLSR` sets `vflag` to 0 or 1. `vflag` is a boolean flag shows whether or not any changes happened in one-hop neighbors of the node and is used for increasing `osn` in TC messages. It means if one-hop neighbors of a node are changed, this value is set to 1; otherwise, it will not be changed. In mobile networks where neighbors change over time, this flag can be used to discover changes, but in our model, the mobility is left as our future work.

- Second, it updates its routing table for the message originator `oip` according to the information existing in HELLO messages by the `update` function. This happens in such a manner - the message `oip` is replaced by `dip` in the routing table, the `hops` sets to 1, `nhopip` changes to the address of message originator, `oip`, `onehop` neighbors are updated, and other elements of the `rt` will not change by receiving HELLO message. After applying the `update` function, nodes know about their immediate neighbors.

- Then by applying the `updatetwohop` function, updating two-hop neighbors is done. It means the `OLSR` checks `onehop` neighbors of the received message originator. If their value is equal to one, the `OLSR` updates the routing table for those nodes. It updates `dip` which sets to the address of selected nodes. The `hops` is updated as follows: *a*) if the `hops` has been updated already and is equal to 1 by applying `update` function, it will not be changed. *b*) if the `hops` has not been updated before, it will set to 2. Finally, the `nhopip` is updated.

validity time of the messages will not be expired.

a) if the nhopip has been updated already using the update function, it will not be changed. *b*) if the nhopip has not been updated before, it will be set by replacing the oip of the message with the nhopip. After that updatetwohop function performed its responsibility, all nodes have the information about their two-hop neighbors.

- Finally, the OLSR deletes the message by deletemsg function since these types of messages cannot be forwarded and then sets idle to 1.

The process of broadcasting HELLO messages happens every 2000 milliseconds. Therefore, the Controller will be activated again as soon as t reaches [4000,4010) milliseconds and all the synchronizations and updates will occur at this time. The process is the same when t reaches [6000, 6010), [8000, 8010), etc milliseconds.

5.2.1.2 Broadcasting TC Messages

To model on-time broadcasting of TC message, we define a constant time_between_tc equals to 5000. When time t reaches [5000,5010), Controller synchronizes with OLSR via CreateTc channel and OLSR creates and sends TC messages at this time. The Controller transits to the next state without any update and the OLSR transits to the next state and resets the clock t to 0 and updates its status to not idle. The OLSR remains in the next state until the transition guard, i.e., t>=time_sending - time_spread, is enabled. If the guard is satisfied, t is in [35, 45] milliseconds and the transition is taken, synchronization by the Queue via channel tc happens and the following actions are occurred. First, the sn[18] is determined either by incrementing by 1 or by last value of sn. When vflag is 1 due to last received HELLO messages, sn is incremented by 1, otherwise is updated by the last value of sn. In other words, if one-hop neighbors of a node has changed, the node has to increment its sn [13]. Second, TC message is created by the OLSR using the createtc function. Every TC is composed of the following elements as described in section 2.5.2:

- msgtype which is TC.
- oip that represents the message originator. The OLSR sets this value by assigning its ip into this element.
- osn shows message sequence number. This value is filled by sn which was determined after first action.
- ttltc indicates time to live of TC message which is equal to N-1.
- hops illustrates the distance from the originator of the message to the receiver node. This value is 0 when the message is created.
- vtime_tc which is set to 15000 milliseconds and is the valid time of TC message.
- onehop is a boolean array of one-hop neighbors of the originator.

[18] The sn was defined earlier in this chapter.

- sip is the sender of the message. First time the OLSR creates a TC message, it assigns its ip to this value. When a TC is forwarded, the sender OLSR copies its ip into this value. When a node receives TC message from the sip, it shows that sip is one hop away.

The msgglobal variable is set to the created TC message and is sent to one-hop neighbors. one-hop neighbors are synchronized by TC sender via tc channel and finally the sender sets the vflag to 0 and becomes idle. The Queue timed automaton as the receiver adds msgglobal to its local messages using the addmsg function, and is synchronized with the OLSR timed automaton when it is idle by imsg channel. It sends the last message to OLSR and deletes it from its local messages. According to the section 2.5.2 When OLSR receives TC message, two different transitions can be taken depending on the following conditions:

1. When a node *a*) receives its own TC message, i.e., msglocal.oip==ip, or *b*) the dsn in the routing table is larger than the osn in the received message which shows the message is not fresh, or *c*) by having the same dsn and osn, the hops of entry in the routing table for the message originator is smaller than the hops in the received message. In this situation, the OLSR drops and deletes the message immediately [13], and becomes idle.

2. If *a*) the originator of the TC message is different than the receiver and the osn of the message is larger than the dsn exists in the routing table, or *b*) "if the receiver node is not the message originator, in case of having the same osn and dsn, if the hops of the message is smaller than the one in the routing table"[19]:

   - The routing table will be updated for the received TC message originator, i.e., msglocal.oip, using the update function as following:

     (a) The rt[msglocal.oip].dip will be filled by msglocal.oip.

     (b) If rt[msglocal.oip].hops has not been updated by receiving HELLO message, the hops in the message +1 is substituted with the hops in the routing table for oip, otherwise it remains unchanged.

     (c) The nhopip is determined with respect to some conditions indicated in the model, Fig. 5.2. *a*) If the rt[msglocal.oip].nhopip has been already updated while receiving HELLO message, it will not be changed. *b*) Otherwise, msglocal.sip is replaced with nhopip in the routing table for oip.

     (d) The osn in the TC message will be replaced with the dsn in the routing table for oip.

     (e) onehop neighbors of the message originator are updated according to the information in the TC message.

     (f) The flag updates as follows *a*) if the osn in the message is greater than rt[msglocal.oip].dsn, the flag is reset to 0. *b*) if the osn of

[19] This does the shortest path calculation. It means in case of having a message with smaller hops, the routing table is updated for the originator of this message.

the message is smaller than the one in in the routing table for `oip`, the flag remains unchanged since the message has been forwarded before.

- `t` will be reset and `idle` status will change to `not idle`.

Finally, two different options might be selected:

- If the `hops` in the message is larger than or equal N -1 and the `ttltc` of the message is equal or smaller than 1 or the message has been forwarded before [13], i.e., `rt[msglocal.oip].flag !=0`, the message will not be forwarded and will be deleted by `deletemsg` function and the `OLSR` becomes `idle`.
- In the interval between [35, 45], If the message can still be transferred to other next nodes and it has not been forwarded before via this node, in other words if the `flag` for the originator of this message is 0, the message `sip` will change to the address of the node which is processing the message, the `hops` will be incremented by 1, the `ttltc` will be decreased by 1 [13] using the `newcreatetc` function. Also, `rt[msglocal.oip].flag` is set to 1 which indicates the TC is forwarded. At last, the message will be forwarded to the `Queue` of `isconnected` nodes via `tc` channel. Then, the message is deleted and `OLSR` becomes `idle`.

When `t` is in [10000, 10010) milliseconds, both HELLO and TC messages should be sent. In this case, HELLO message has the higher priority than TC message. Therefore, the `Controller` synchronizes with the `OLSR` on the `CreateHello` channel first, and then on the `CreateTc` channel. The `Controller` also resets its `t` and goes back to the initial state. The whole process repeats as soon as `t` reaches [2000,2010), again.

5.2.2 Same Starting Time

Another situation is when all nodes start working at the exactly same time. Therefore, they broadcast their messages all at the same time. In this case, having only one `Controller` is sufficient for on-time broadcasting. Fig. 5.4 demonstrates the system with only one `Controller` which synchronizes with all `OLSR`s all together and its timing is associated with the global `clk`. All nodes start working when `clk` is equal to 0 and as soon as it reaches 2000 milliseconds, the `Controller` sends signal to all `OLSR`s via the `CreateHello` channel. All `OLSR`s synchronize with the `Controller` and try to create their HELLO messages and update their routing tables as described in section 5.2.1. The process of creating, sending and forwarding TC messages, and updating routing tables is also the same as which was explained in 5.2.1. The only differences are the `Controller` and its synchronization channels, `CreateHello` and `CreateTc`, and the `clk`.

At time 4000, 5000, 6000, 8000 and 10000, the `Controller` is again activated and synchronizes with `OLSR`s. Finally, it resets the global clock `clk`. The `OLSR` with only one `Controller` is illustrated in Fig. 5.5.

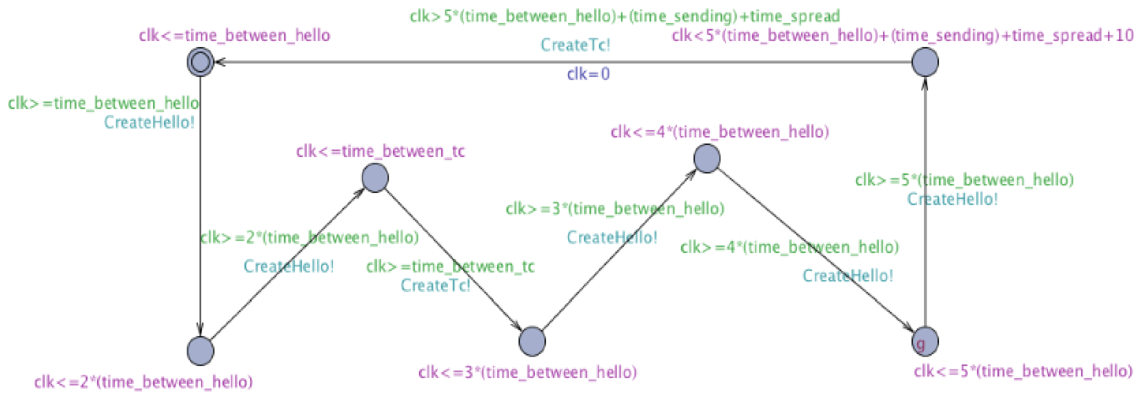
clk>5*(time_between_hello)+(time_sending)+time_spread
clk<=time_between_hello
CreateTc!
clk<5*(time_between_hello)+(time_sending)+time_spread+10
clk=0
clk>=time_between_hello
CreateHello!
clk<=4*(time_between_hello)
clk<=time_between_tc
clk>=5*(time_between_hello)
CreateHello!
clk>=3*(time_between_hello)
clk>=2*(time_between_hello)
CreateHello!
clk>=4*(time_between_hello)
CreateHello!
clk>=time_between_tc
CreateTc!
CreateHello!
g
clk<=2*(time_between_hello)
clk<=3*(time_between_hello)
clk<=5*(time_between_hello)

Fig. 5.4. Same starting time `Controller`.

In order to avoid the state space explosion, we validate and verify our model for the system consisting one `Controller` and make the assumption that all nodes start working at the same time. The advantages of having one `Controller` are: *a*) decreasing number of clocks and states in the system that reduces the state space explosion problem. *b*) ability to reset the global clock `clk` which again helps to reduce the state space explosion. Starting at the same time do not have any effect on the behavior of the nodes. Therefore, the assumption that nodes start working at the same time is considered as a correct assumption.

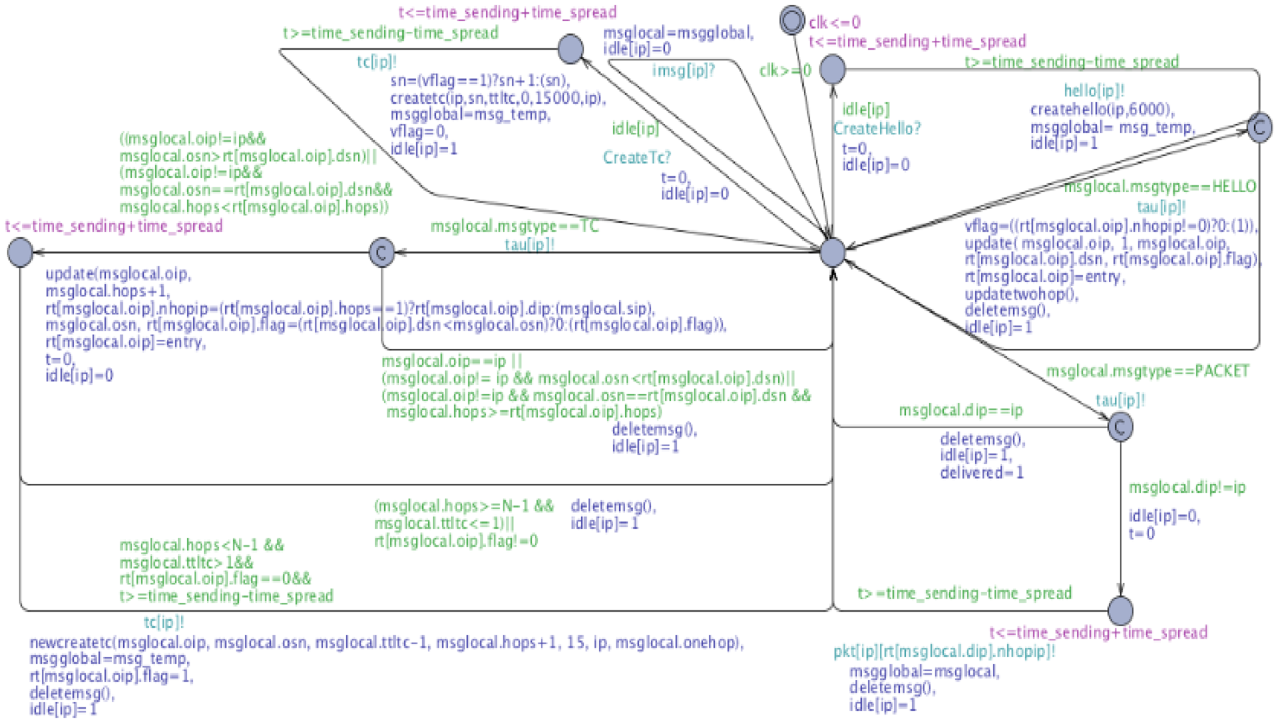
t<=time_sending+time_spread
t>=time_sending-time_spread
msglocal=msgglobal,
idle[ip]=0
clk<=0
t<=time_sending+time_spread
tc[ip]!
imsg[ip]?
clk>=0
t>=time_sending-time_spread
sn=(vflag==1)?sn+1:(sn),
createtc(ip,sn,ttltc,0,15000,ip),
msgglobal=msg_temp,
vflag=0,
idle[ip]=1
idle[ip]
idle[ip]
CreateHello?
t=0,
idle[ip]=0
hello[ip]!
createhello(ip,6000),
msgglobal= msg_temp,
idle[ip]=1
C
((msglocal.oip!=ip&&
msglocal.osn>rt[msglocal.oip].dsn)||
(msglocal.oip!=ip&&
msglocal.osn==rt[msglocal.oip].dsn&&
msglocal.hops<rt[msglocal.oip].hops))
CreateTc?
t=0,
idle[ip]=0
msglocal.msgtype==HELLO
tau[ip]!
t<=time_sending+time_spread
msglocal.msgtype==TC
tau[ip]!
C
vflag=((rt[msglocal.oip].nhopip!=0)?0:(1)),
update(msglocal.oip, 1, msglocal.oip,
rt[msglocal.oip].dsn, rt[msglocal.oip].flag),
rt[msglocal.oip]=entry,
updatetwohop(),
deletemsg(),
idle[ip]=1
update(msglocal.oip,
msglocal.hops+1,
rt[msglocal.oip].nhopip=(rt[msglocal.oip].hops==1)?rt[msglocal.oip].dip:(msglocal.sip),
msglocal.osn, rt[msglocal.oip].flag=(rt[msglocal.oip].dsn<msglocal.osn)?0:(rt[msglocal.oip].flag)),
rt[msglocal.oip]=entry,
t=0,
idle[ip]=0
msglocal.oip==ip ||
(msglocal.oip!= ip && msglocal.osn<rt[msglocal.oip].dsn)||
(msglocal.oip!=ip && msglocal.osn==rt[msglocal.oip].dsn &&
msglocal.hops>=rt[msglocal.oip].hops)
deletemsg(),
idle[ip]=1
msglocal.msgtype==PACKET
tau[ip]!
msglocal.dip==ip
C
deletemsg(),
idle[ip]=1,
delivered=1
msglocal.dip!=ip
idle[ip]=0,
t=0
(msglocal.hops>=N-1 &&
msglocal.ttltc<=1)||
rt[msglocal.oip].flag!=0
deletemsg(),
idle[ip]=1
msglocal.hops<N-1 &&
msglocal.ttltc>1&&
rt[msglocal.oip].flag==0&&
t>=time_sending-time_spread
t>=time_sending-time_spread
tc[ip]!
t<=time_sending+time_spread
newcreatetc(msglocal.oip, msglocal.osn, msglocal.ttltc-1, msglocal.hops+1, 15, ip, msglocal.onehop),
msgglobal=msg_temp,
rt[msglocal.oip].flag=1,
deletemsg(),
idle[ip]=1
pkt[ip][rt[msglocal.dip].nhopip]!
msgglobal=msglocal,
deletemsg(),
idle[ip]=1

Fig. 5.5. `OLSR` with one `Controller`.

5.3 Model Discussion

We explained our model in previous sections. In this section, we briefly discuss about 2 parts of our model:

- The reasons for having 3 timed automata; the `Controller`, the `Queue`, and the `OLSR`.
- The way of defining synchronization channels.

As we illustrated, we modeled every node as a parallel composition of three timed automata to have a model which reflects *a*) precise timing, *b*) an organized process for message handling, *c*) nodes behavior.

The `Controller` is required to be a separate automaton to serve as a manager for nodes to broadcast their messages at certain times. As a consequence, we had to define this automaton for the system, separately. The `Queue` as the next automaton is needed to store incoming messages from other nodes. This automaton helps to sort out these incoming messages based on their reception and also allows the `OLSR` automaton to process messages in turn. Also, the `Queue` is responsible to create packets if needed. Clearly, having this automaton is considered as a requirement for our model. The `OLSR` automaton definitely is the most important element of our system, since it reflects the behavior of nodes.

These different automata can communicate with each other using synchronization channels. As described in section 3.2, binary and broadcast synchronization allow us to model unicast and broadcast communications of the OLSR protocol. In reality, the OLSR broadcasts its control messages in the entire network. As the result, we have used broadcast channels to send these messages, channels `hello` and `tc`. For the purpose of `Controller` and `OLSR` synchronization, we have applied two other channels `CreateTc` and `CreateHello`, as discussed in section 5.2. These channels are broadcast channels, since the `Controller` automaton have to synchronize with some other automata, all `OLSR`s, when nodes start working at the same time.

When injecting a packet at some point, only the `Tester` and one `Queue` have to be synchronized which is an unicast synchronization. As a consequence, we defined an urgent unicast channel known as `packet`. This channel is defined as an Urgent channel, since the process of injecting a packet has to be happened with no delay. Channel `pkt` was modeled as an unicast channel, because only two automata are synchronizing, the `OLSR` and the `Queue`. We also need one channel to transfer messages from the `Queue` to the `OLSR`. So, we defined an urgent unicast channel called `imsg`. Since only two automata `Queue` and `OLSR` have to be synchronized without any delay, we need an urgent unicast channel. As mentioned in section 5.2, the purpose of selecting channel `tau` is decreasing the state space explosion. When a message is transferred from the `Queue` to the `OLSR`, this automaton takes the transition which labeled by `tau` channel immediately and then starts processing according to the type of the message. Therefore, this channel is denoted as an urgent broadcast channel.

5.4 Topology Discussion

We described in section 5.1 that fundamental flaws can be emerged in small networks. For instance, if nodes cannot find routes to other nodes, they cannot deliver their data packets to different destinations. Moreover, disability of finding optimal routes might happen in small networks as well [11]. Considering the fact that these fundamental flaws even can be appeared in small topologies and our goal in this study is verification of such properties, we "verify" our model for all networks of 3 and 4 nodes. The restriction to these small networks is also to avoid state space explosion. Definitely, verification of these and other properties in larger networks is left as our future work.

Two examples of such topologies are shown in Fig. 5.6 and 5.7.

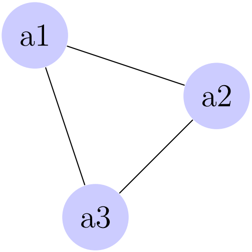

Fig. 5.6. 3 nodes topology.

We investigate the "validation" of our work by the simulator in Uppaal for these two sample networks in chapter 6.

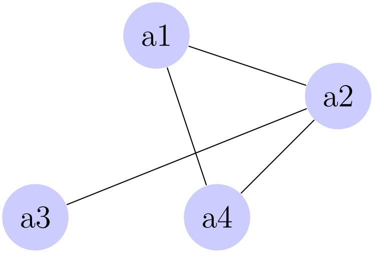

Fig. 5.7. 4 nodes topology.

Chapter 6

Experiments

In this chapter, we validate and verify our Uppaal model using simulator and model checker. For these, we define an automaton called `Tester`. We run our simulator for network topologies depicted in Fig. 5.6 and 5.7 for model validation when sending HELLO, TC and PACKET. In the verification part, we verify our model for required properties such as:

- Finding optimal routes
- Packet delivery

In our model for all `(i:int[0,N-1])`, `ai` represents the `OLSR(i)` automaton and `aiq` indicates the `Queue(i)` automaton.

6.1 Modeling Tester (injected packet)

The `Tester` is used for the purpose of checking packet delivery. This automaton comprises 3 states and 2 transitions. This automaton can be activated when the `clk` reaches 7000 milliseconds by considering the invariant on the initial state and the guard on the transition. After taking the transition, the state of the system will change and the automaton will move to the next state. On the next state of the process, `Tester` and the `Queue` will synchronize on `packet` channel aiming at sending a packet from `OIP1` that is constant equal to 1 which is the originator of packet and `DIP1` that is also a constant equal to 3, the destination node. It means that the packet delivery process is done if a packet is injected to node `a1` will be delivered to the destination which is node `a3`. Fig. 6.1 represents this automaton. When the `Queue` of `isconnected` nodes synchronize with the `Tester`, they will start to create the packet by the `createpacket` function and add the packet into the local messages of the `Queue` by the `addmsg` function. The packet contains:

- `msgtype` which is PACKET.
- `oip` shows the address of the originator.
- `dip` represents the destination node.

In next step, the Queue and the OLSR synchronize on imsg channel by passing the packet from the Queue to the OLSR for processing. In this step, two guards are checked:

- If the destination of the packet is the node which received the packet, the message will be deleted, idle will be set to 1 and delivered which is a boolean value shows if the packet has been delivered to the destination will be set to 1.[1]

- If the destination of the packet is different from the node which received the packet, the OLSR becomes busy and t will be reset to 0 and the OLSR has to wait on the next state until the guard is satisfied. Then as soon as t reaches the time between [35, 45], the packet will be sent to the Queue of next isconnected node along the path to the destination with respect to nhopip in the routing table via pkt channel. The OLSR copies the msglocal or the packet into msgglobal. At last, after sending the packet to the next node, the message will be deleted and the OLSR becomes idle. The packet is sent to next nodes until the destination is reached. Finally the destination, lets say node a3 which received the injected packet, sets its boolean delivered to 1.

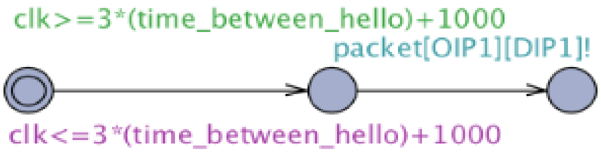
clk>=3*(time_between_hello)+1000
packet[OIP1][DIP1]!
clk<=3*(time_between_hello)+1000

Fig. 6.1. Tester automaton.

6.2 Validation

Validation refers to a dynamic process of checking and testing if the constructed model fulfills its intended use. It contains executing the code and ensures whether or not a right model is built.

6.2.1 Simulator

The dynamic behavior of nodes can be indicated by using simulator in Uppaal. Fig. 6.2 shows the HELLO message sequence chart returned by Uppaal. It illustrates

[1] To check packet delivery, we define a boolean delivered for every OLSR automaton. When the destination node receives the packet, it sets this value to 1.

the activities and messages sent by the three nodes and the `Controller`. The topology of nodes is depicted in Fig. 5.6. As we mentioned in section 5.2, the connectivity between nodes is determined by `isconneted` predicate and the network topology is determined by a matrix called topology which is a two-dimensional array of size N. We have used

$$\texttt{topology}[\texttt{N}][\texttt{N}] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

as the topology to run our simulator. The scenario begins with the `Controller` to inform nodes for broadcasting HELLO messages. Once this has occurred, node `a1`, `a2` and `a3` broadcast their messages which are received by the `Queue` of node `a2` and `a3`, `a1` and `a3`, `a1` and `a2`, respectively. These activities occur every 2000 milliseconds by all nodes in the network. The message sequence chart displays only the channels name that were used, in here the broadcast channel `hello`. Moreover, Uppaal reports the variables values at every node, but this information is not included in the chart.

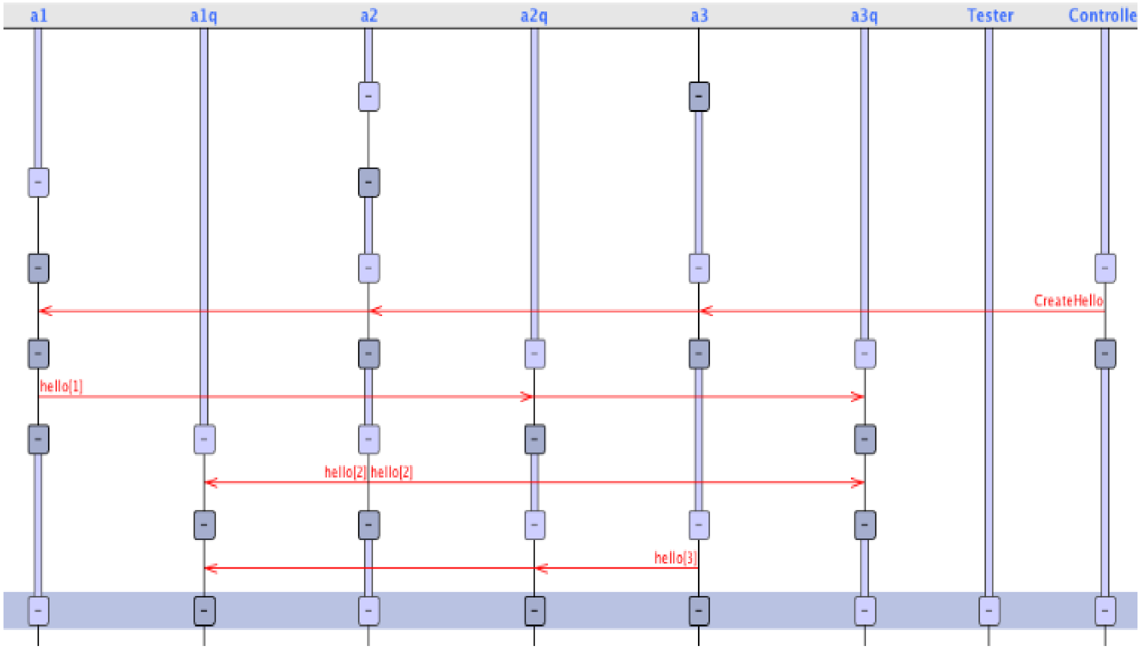

Fig. 6.2. Broadcasting HELLO message sequence chart.

Updated routing table for node `a1` after sending and receiving first and second HELLO messages when `clk`=5000 is depicted in table 6.1.

dip	hops	nhopip	dsn	flag	onehop
0					
1					
2	1	2			1,3
3	1	3			1,2

Table 6.1. Updated routing table for node `a1` at `clk`=5000.

Fig. 6.3 demonstrates the activities of the `Controller` and nodes broadcasting TC messages. In this figure, all 3 nodes broadcast TC messages to the `Queue` of their `isconnected` nodes which happens every 5000 milliseconds. After sending, receiving and processing TC messages by nodes, forwarding those messages will be done which is depicted in Fig. 6.4.

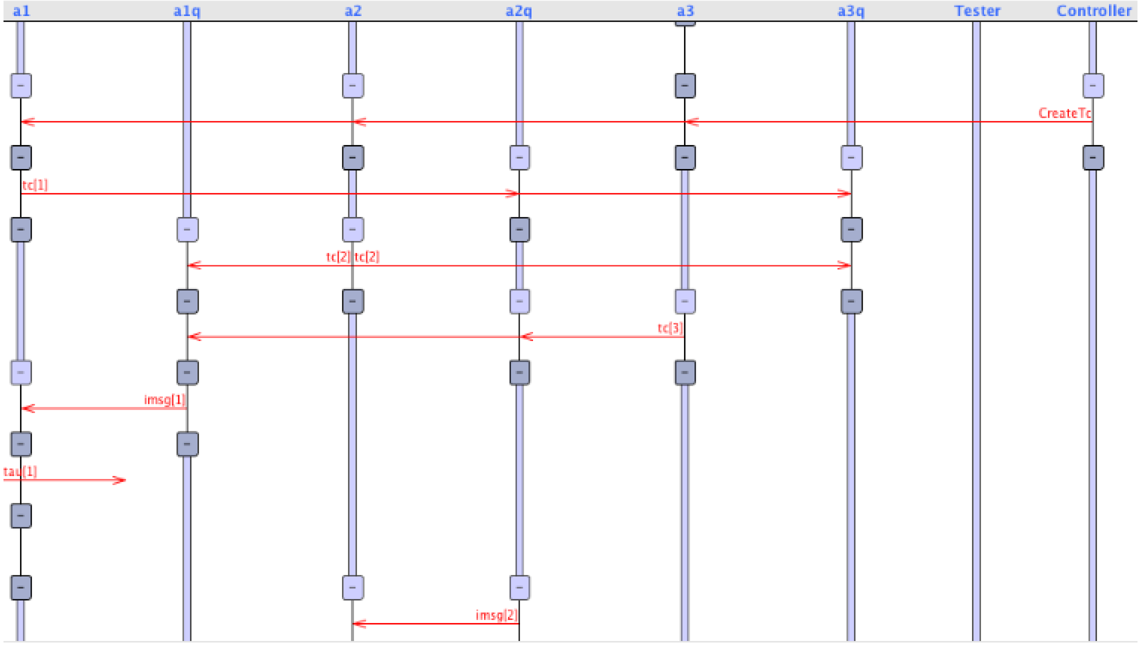

Fig. 6.3. Broadcasting TC message sequence chart.

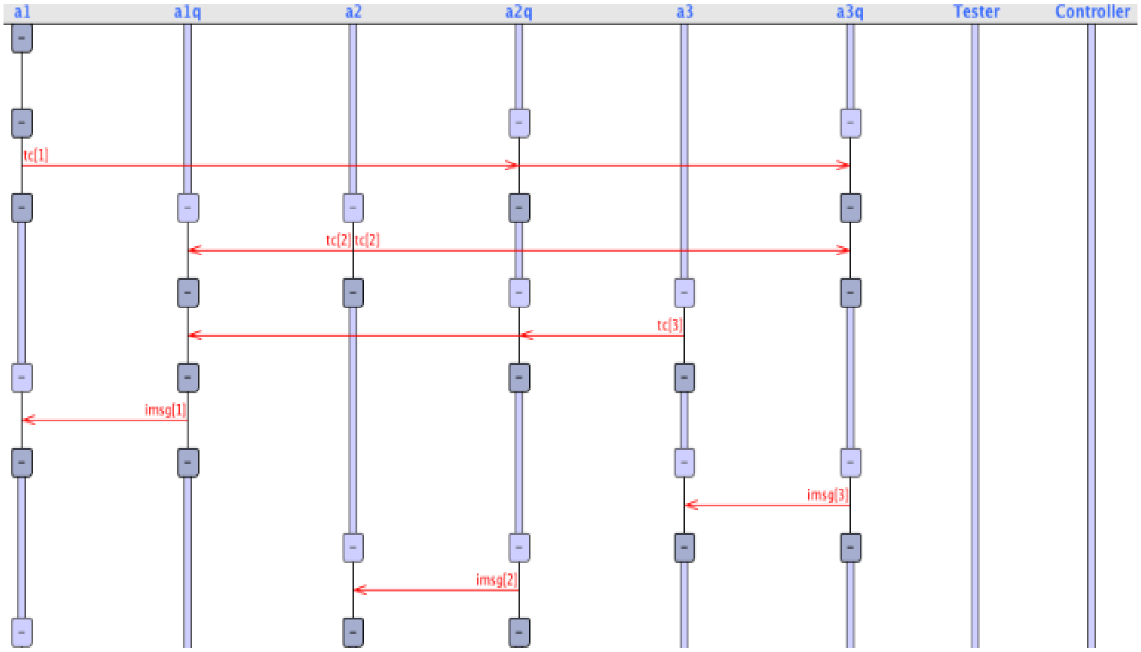

Fig. 6.4. Forwarding TC message sequence chart.

Table 6.2 demonstrates the updated routing table for node `a1` after receiving TC messages by node `a2` and `a3` at `clk`=6000.

dip	hops	nhopip	dsn	flag	onehop
0					
1					
2	1	2	1	1	1,3
3	1	3	1	1	1,2

Table 6.2. Updated routing table for node `a1` at `clk`=6000 having 3 nodes.

When `clk` reaches 7000 milliseconds, it trie to send the packet from constant `OIP1`=1, i.e., packet originator, to constant `DIP1`=3, i.e., destination node. This process is given in Fig. 6.5. The simulator indicates that injected packet is delivered at the destination, node `a3`, since `a3.delivered=1`.

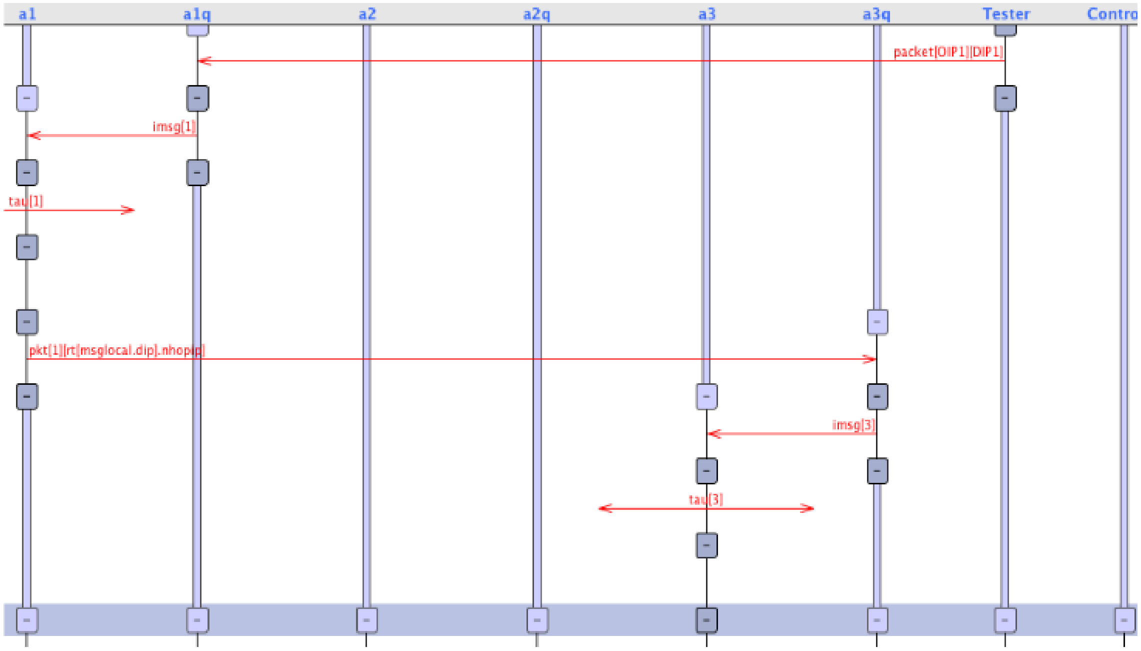

Fig. 6.5. Injecting packet sequence chart for 3 nodes.

In addition, we run our simulator for another topology with 4 nodes equipped in a network as shown in Fig. 5.7. The following matrix depicts the connectivity between these 4 nodes and the numbers in the matrix show distances/ hops from every node to other nodes in the network.

$$\texttt{topology}[\texttt{N}][\texttt{N}] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 2 & 1 & 0 & 2 \\ 0 & 1 & 1 & 2 & 0 \end{bmatrix}$$

We represent the routing table for node `a1` after passing 6000 milliseconds. At this time, all nodes have broadcasted their HELLO and TC messages, so they have all the required information for routing a packet. Table 6.3 contains this information.

dip	hops	nhopip	dsn	flag	onehop
0					
1					
2	1	2	1	1	1,3,4
3	2	2	1	1	2
4	1	4	1	1	1,2

Table 6.3. Updated routing table for node `a1` at `clk`=6000 having 4 nodes.

In this topology we inject a packet at node `a1` to be received at node `a3`. As table 6.3 demonstrates the packet has to cross two hops to reach node `a3`. Therefore, the packet is sent to the next node along the path to the destination, i.e., node a3. This table represents that the next node is node a2. So, the packet is sent to node a2. The routing table for node a2 is given in table 6.4. Node a2 then uses its routing table to find the next node to node a3. Finally, node a2 sends the packet to node a3 and this node changes its boolean delivered status to 1, i.e., a3.delivered=1, that shows the destination node, a3 has received the packet. Fig. 6.6 indicates this sequence chart of sending packet for 4 nodes.

dip	hops	nhopip	dsn	flag	onehop
0					
1	1	1	1	1	4,2
2					
3	1	3	1	1	2
4	1	4	1	1	1,2

Table 6.4. Updated routing table for node a2 at clk=6000 having 4 nodes.

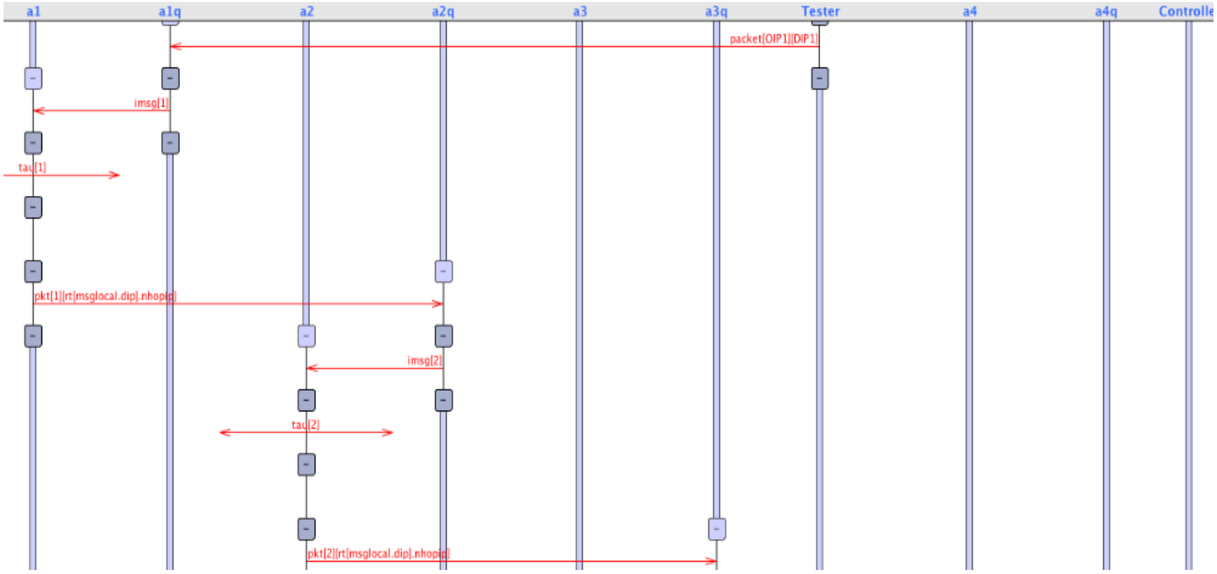

Fig. 6.6. Injecting packet sequence chart for 4 nodes.

6.3 Verification

The main purpose of using the Uppaal verifier is verifying our OLSR model with respect to a requirement specification. Like our OLSR model, the requirement specification must be represented as a formally machine readable and well-defined language. As mentioned in section 3.4, Uppaal uses CTL to state queries.

Our verification experiments split into two categories: optimal route establishment and packet delivery. Since we have modeled our system in a way that works in every network, we examine these two properties for different topologies up to 4 nodes. As we stated before in section 5.4, if there is any fundamental flaws in the system, it will be appeared in even these small topologies. We translate these queries in CTL syntax and verify them for our OLSR model.

6.3.1 Optimal Route Establishment

As we mentioned in section 2.5.3, OLSR seeks to find the shortest path from the source node to the destination node. For this reason, we use the Uppaal verifier.

We explained in section 6.2.1 that the numbers in `topology` matrix represent the number of hops between two nodes. Therefore, we can easily use `topology[i][j]` to check how many hops the difference between two nodes is. `i` and `j` are the addresses of two nodes in the network `(i,j:int[0,N-1])`. In addition, the number of hops in the routing table from one node to the another node is defined as `ai.rt[j].hops`. If these two values are equal, it shows that the optimal route is found; otherwise, non-optimal route is found.

As a consequence, the property required to be verified for finding optimal routes using CTL is expressed as:

$$A \texttt{<> (ai.rt[j].hops==topology[i][j])}$$

saying that for all paths finally[2] the `hops` between two nodes in the routing table is equal to the matching number in the `topology` matrix.

We verified this property for different topologies and for different nodes in the network. We have examined our model for networks of 3 and 4 nodes. The property got satisfied for all networks up to 4 nodes. This shows all nodes can find the best optimal route to other nodes in the network.

6.3.2 Packet Delivery

Another property which is required to be verified is packet delivery property. This property shows if a packet injected into the network is delivered at the destination using the information in the routing tables. We check this property in a way that injected packet at `clk`=7000 by `OIP1` that is a constant equal to 1, will always be delivered at the destination `DIP1` which is also a constant equal to 3.[3] As explained above in section 6.1, “if the boolean `delivered` of `DIP1` is equal to 1”,[4] the packet has been received by the destination. We check the property which says for all paths when the `Controller` is in state `g`, i.e., `clk`=10000, the packet is finally received by node `a3`. Clearly when the packet has been sent at 7000 milliseconds, even by considering the delay and time consumed for sending the packet, it must be delivered when the time of the system is 10000. In Uppaal, this property is illustrated as :

[2] Path formula `<>` ϕ represents ϕ holds eventually in some state of a path.

[3] We inject a packet into the system when the global clock `clk` is reached 7000 milliseconds. At this time HELLO and TC messages have been sent already and the required information for routing the packet is provided in routing tables.

[4] `a3.delivered==1`

A `<>(Controller.g imply a3.delivered==1)`

`Controller.g` is the state where the `clk` is equal to 10000. By adding `Controller.g` to the formula, we restrict Uppaal to only search the states when `clk<=10000` where there is the possibility for the packet to be delivered at the destination. Otherwise, Uppaal has indeed to check all reachable paths at all times which we really do not need and yields state space explosion. We also verified this property for all networks consisting 3 and 4 nodes. The property for all these networks got satisfied. To put the subject into a nutshell, it can be asserted that OLSR is able to find optimal routes between nodes to deliver data packets.

6.4 Results

We implemented a formal specification of the OLSR core functionality which is derived from the RFC 3626. This formal model allows to have only one specific interpretation from this protocol. We validated our model using the Uppaal simulator earlier in this chapter. Also, by means of verification part of the Uppaal, we verified two important properties. First, we translated these two properties using CTL to be understandable by Uppaal verifier.

We verified that all nodes can establish optimal routes to other nodes in the network. In addition, we verified that the injected packet by the user layer has been delivered at the destination which shows nodes have the information about the whole network after broadcasting control messages.

We verified these two properties for all networks of 3 and 4 nodes.[5] For these small networks, analyzing all topologies for given properties is possible which gives a good overview of the behavior in different situations. Moreover, this systematic analysis of small networks helps to select the most informative scenario for larger networks in our future work.

[5] In general, we investigated 55 different topologies.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In section 1.1, we sketched the problem of English specifications for different routing algorithms used in WMNs, which leads to ambiguities or different interpretations. Since all people have the same interpretations from mathematical formulas, formal modeling is considered as an approach to remove ambiguities. Our purpose in this study was twofold: (i) We modeled a formal and unambiguous model of OLSR and its main functionalities using timed automata as our formal specification language in order to carry out some systematic analysis across all small networks. (ii) By a careful analysis of OLSR using Uppaal verifier, we verified two important properties for our model. These properties are having a system enabled to find optimal routes, and deliver a packet to another node. Our work can be considered as a complementary to traditional techniques such as simulations and test-bed experiences. Since OLSR is still under development, we provide valuable feedback[1] to the developers, who might take our findings into account when generating the final specification of OLSR.

7.2 Future Work

One of our purposes for the future is extending the model for other functionality of the OLSR and also investigating and analyzing systems with both MPR and not MPR nodes. We also plan to optimize our model to cover all topologies up to 5 or 6 nodes. In future work, we also extend a number of mobility models to find out the behavior of this routing protocol. Moreover, we are going to compare OLSR with other WMN protocols. For instance, one attempt might be verifying the results of [34]. Another plan might be using the model in larger network topologies and verifying properties for those large networks by Statistical Model Checking (SMC).

[1] We verified these two important properties. As a consequence, it can be concluded that OLSR is able to deliver data packets to other nodes via optimal routes in all investigated networks up to 4 nodes.

Bibliography

[1] "Greenorbs: A long-term kilo-scale wireless sensor network system in the forest," http://www.greenorbs.org/all/greenorbs.htm, 2009, accessed: 2014-05-07.

[2] "Mesh network monitors volcanoes," http://www.technologyreview.com/view/413969/mesh-network-monitors-volcanoes/, 2009, accessed: 2014-05-07.

[3] "Security mechanisms in wireless mesh networks (wmns)," http://www.cs.iastate.edu/~jxiawang/research.htm, 2007, accessed: 2014-05-07.

[4] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: A survey," *Comput. Netw. ISDN Syst.*, vol. 47, no. 4, pp. 445–487, Mar. 2005.

[5] "Rapid deployment solutions," http://www.efjohnsontechnologies.com/resources/dyn/files/75463z3745f320/_fn/RapidDeploymentSolutions.pdf, accessed: 2014-05-07.

[6] S. Edenhofer and P. Höfner, "Towards a rigorous analysis of AODVv2 (DYMO)," *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–6, 2012.

[7] A. Fehnker, R. Van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan, "A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV," *arXiv preprint arXiv:1312.7645*, 2013.

[8] G. Chua and R. Patterson, "WiFi mesh private market overview, market analysis, IDC," April 2008.

[9] L. A. Frost and D. L. Sullivan, "World mobile backhaul infrastructure market, market analysis," 2009.

[10] P. Höfner, Personal communication, peter.Hoefner@nicta.com.au.

[11] A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan, "Modelling and analysis of AODV in UPPAAL," in *1st International Workshop on Rigorous Protocol Engineering*, Vancouver, October 2011, pp. 1–6.

[12] A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan, "Automated analysis of AODV using UPPAAL," in *18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012).* Tallinn, Estonia: Springer, March 2012, pp. 173–187.

[13] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)," RFC 3626 (Experimental), Internet Engineering Task Force, October 2003.

[14] K. R. KRISHNAIAH[1], B. R. BABU, T. P. KUMAR, and K. R. RAO, "Wireless mesh networks-reliability and flexibility," vol. 2, no. 1, pp. 23–29, May 2009.

[15] T. Lei, F. Long, R. Barzilay, and M. C. Rinard, "From natural language specifications to program input parsers," in *ACL (1)*, August 2013, pp. 1294–1303.

[16] R. van Glabbeek, P. Höfner, M. Portmann, and W. L. Tan, "Sequence numbers do not guarantee loop freedom —AODV can yield routing loops—," in *Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'13).* ACM, 2013, pp. 91–100.

[17] K. Mase, Y. Owada, H. Okada, and T. Imai, "A testbed-based approach to develop layer 3 wireless mesh network protocols," in *Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks Communities*, ser. TridentCom 08. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 30:1–30:6.

[18] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized Link State Routing Protocol for Ad Hoc Networks," in *Multi Topic Conference, 2001. IEEE INMIC 2001.* IEEE, 2001, pp. 62 – 68.

[19] "Dijkstra's algorithm — Wikipedia, the free encyclopedia," [accessed 08-May-2014]. [Online]. Available: http://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFDijkstra1959

[20] M. Kamali, *Reusable Formal Architectures for Networked Systems*, ser. TUCS Dissertations. Turku centre for computer science, 2013. [Online]. Available: http://books.google.fi/books?id=0CI7ngEACAAJ

[21] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[22] A. Mclver and A. Fehnker, "Formal techniques for the analysis of wireless networks," in *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on.* IEEE, 2006, pp. 263–270.

[23] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking.* Cambridge, MA, USA: MIT Press, 1999.

[24] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[25] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on Uppaal," in *SFM*, 2004, pp. 200–236.

[26] E. M. Clarke, E. A. Emerson, and J. Sifakis, “Model checking: Algorithmic verification and debugging,” *Commun. ACM*, vol. 52, no. 11, pp. 74–84, 2009.

[27] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan, “A process algebra for wireless mesh networks.” in *ESOP*, ser. Lecture Notes in Computer Science, vol. 7211. Springer, 2012, pp. 295–315.

[28] K. G. Larsen, P. Pettersson, and W. Yi, “Uppaal in a nutshell,” *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, May 1997.

[29] E. A. Emerson, “Temporal and modal logic,” in *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*, 1995, pp. 995–1072.

[30] S. Chiyangwa and M. Z. Kwiatkowska, “A timing analysis of AODV.” in *FMOODS*, ser. Lecture Notes in Computer Science, vol. 3535. Springer, 2005, pp. 306–321.

[31] P. Höfner and A. McIver, “Statistical model checking of wireless mesh routing protocols,” in *5th NASA Formal Methods Symposium (NFM 2013)*, N. R. G. Brat and A. Venet, Eds., vol. 7871. Moffett Field, CA, USA: Springer, May 2013, pp. 322–336.

[32] M. F. Steele and T. R. Andel, “Modeling the optimized link-state routing protocol for verification,” in *SpringSim (TMS-DEVS)*. Society for Computer Simulation International, 2012, pp. 35:1–35:8.

[33] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing,” RFC 3561 (Experimental), Internet Engineering Task Force, July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[34] A. Huhtonen, “Comparing AODV and OLSR routing protocols,” in *Seminar on Internetworking, Sjkulla*, 2004, pp. 26–27.

Appendices

Appendix A

Functions

A.1 OLSR automaton functions

```
void deletemsg(){
    msglocal.msgtype=NONE;
    }
    void createhello(IP oip, VTIME_H vtime_h){
    MSG msg;
    msg.msgtype= HELLO;
    msg.oip= oip;
    msg.vtime_h= vtime_h;
    for(i:int[1,N-1]){
    if (rt[i].hops==1){
    msg.onehop[i] = 1; }
    else {msg.onehop[i]= 0;} }
    msg_temp= msg;
    }
    void createtc(IP oip, SQN osn, int ttltc, int hops, VTIME_TC vtime_tc, IP sip){
    MSG msg;
    msg.msgtype= TC;
    msg.oip= oip;
    msg.osn= osn; // originator of TC SQN
    msg.ttltc= ttltc;
    msg.hops= hops;
    msg.vtime_tc=vtime_tc;
    // create onehop list
    //———————————
    for(i:int[1,N-1]){
    if (rt[i].hops==1){
    msg.onehop[i] = 1;
    }
    else {msg.onehop[i]= 0;}
}
//——————————
msg.sip= sip;
msg_temp= msg;
}
void newcreatetc(IP oip, SQN osn, int ttltc, int hops, VTIME_TC vtime_tc,IP sip,
bool onehop){
MSG msg;
msg.msgtype= TC;
msg.oip= oip;
msg.osn= osn; // originator of TC SQN
msg.ttltc= ttltc;
msg.hops= hops;
msg.vtime_tc=vtime_tc;
msg.onehop=onehop;
msg.sip= sip;
msg_temp= msg;
}
void update(IP dip , int hops, IP nhopip, SQN dsn, bool flag){
rtentry newentry;
newentry.dip=dip;
newentry.hops=hops;
newentry.nhopip=nhopip;
newentry.dsn=dsn;
newentry.flag=flag;
for(i:int[1,N-1]){
if (msglocal.onehop[i]==1){
newentry.onehop[i] = 1;}
else {newentry.onehop[i]= 0;}
entry=newentry; }
}
void updatetwohop(){
for(i:int[1,N-1]){
if(msglocal.onehop[i]==1 && i!=ip){
rt[i].dip=i;
rt[i].hops= (rt[i].hops==1)?1:2;
rt[i].nhopip= (rt[i].hops==1)?rt[i].dip:(msglocal.oip); } }
}
```

A.2 Queue automaton functions

```
void addmsg(MSG msg){
    msglocal[nodebuffersize[ip]]=msg;
    nodebuffersize[ip]++;
    }
    void deletemsg(){
    MSG msg;
    for(i: int[1,M-1]){
    msglocal[i-1]=msglocal[i];
    } msglocal[M-1]=msg;
    nodebuffersize[ip]- -;
    }
    MSGTYPE nextmsg(){
    return msglocal[0].msgtype;
    }
    void createpacket(IP oip, IP dip){
    MSG msg;
    msg.msgtype= PACKET;
    msg.oip= oip;
    msg.dip=dip;
    Packet=msg;
    }
```