ÉCOLE POLYTECHNIQUE ParisTech

Thèse présentée pour obtenir le grade de

DOCTEUR DE L'ECOLE POLYTECHNIQUE

en Mathématiques et Informatique

par

Ulrich HERBERG

Réseaux Ad Hoc : Performance, Dimensionnement, Gestion Automatisée et Intégration dans l'Internet

Soutenue le 20 mai 2011 devant le jury composé de:

Michel Diaz	LAAS-CNRS	Rapporteur
Joseph Macker	Naval Research Laboratory	Rapporteur
Kenichi Mase	Université de Niigata	Rapporteur
Robert G. Cole	US Army	Examinateur
Walid Dabbous	INRIA	Examinateur
Christopher Dearlove	BAE Systems	Examinateur
Philippe Jacquet	INRIA	Directeur de Thèse
Thomas Heide Clausen	Ecole Polytechnique	Co-encadrant

Thesis for the degree of

DOCTEUR DE L'ECOLE POLYTECHNIQUE

in Mathematics and Computer Science

handed in by

Ulrich Herberg

Performance, Scalability, Automatic Management and Internet Integration of Ad Hoc Networks

Defended on May 20, 2011

Abstract

This manuscript provides several ad hoc routing protocol extensions and evaluations thereof, that allow increasing performance, scaling the network size, and integrating ad hoc networks into the Internet:

First, this manuscript presents software tools, which allow to study unmodified routing protocols, implemented in Java, in the network simulator NS2.

Second, the manuscript provides an architectural discussion of ad hoc networks, which explains the problems of automatic configuration of IP addresses (called "autoconfiguration") on ad hoc routers, and presents an architectural model for ad hoc networks, compatible with the Internet. An autoconfiguration protocol, based on the proposed architectural model, is specified, and its properties are formally verified by means of model checking.

Third, several extensions to the Optimized Link State Routing Protocol version 2 (OLSRv2) are presented: The in-router performance of the protocol is largely increased when using a dynamic shortest path algorithm for calculating routes.

Another proposed optimization is an SNMP-based framework to manage and control performance related objects on routers running OLSRv2.

An extension of OLSRv2 is presented, where packets are retransmitted later for temporarily unavailable destinations, instead of dropping them, leading to delivery ratios significantly higher than standard OLSRv2 in certain scenarios.

This manuscript presents an analysis of security threats to OLSRv2, followed by a specification of a router and link admittance control mechanism for OLSRv2, alleviating many of these attacks.

Fourth, this manuscript investigates scalability of wireless sensor networks (WSNs), and evaluates a routing protocol for such WSNs being specified by the IETF as of 2011, denoted RPL. Moreover, as RPL does not contain any mechanism for efficient broadcasts, several optimized broadcasting mechanisms in RPL are presented and compared.

Résumé

Le présent rapport fournit plusieurs extensions des protocoles de réseau ad hoc et leurs évaluations, qui permettent d'augmenter la performance, de dimensionner la taille des réseaux et d'intégrer les réseaux ad hoc dans l'Internet :

Premièrement, ce rapport présente plusieurs outils de logiciel qui permettent d'étudier des protocoles de routage sans modifications, implémentés en Java, dans le simulator de réseaux NS2.

Deuxièmement, ce rapport fournit une discussion architecturale des réseaux ad hoc, qui explique les problèmes de la configuration automatique des adresses IP (nommé ≪ autoconfiguration ≫) des routeurs ad hoc. Ensuite, le rapport présente un modèle architectural pour des réseaux ad hoc qui est compatible avec l'Internet. Un protocole d'autoconfiguration, basé sur le modèle architectural proposé, est spécifié, et ses propriétés sont formellement vérifiées au moyen de ≪ Model Checking ≫.

Troisièmement, plusieurs extensions du protocole de routage des réseaux ad hoc OLSRv2 (≪ Optimized Link State Routing Protocol version 2 ≫) sont présentées : La performance du protocole est augmentée d'ordre de grandeur en utilisant un algorithme dynamique pour calculer les plus courts chemins.

Une autre optimisation proposée est un framework basé sur SNMP pour gérer et contrôler des objets liés à la performance sur des routeurs tournant OLSRv2.

Une extension d'OLSRv2 est présentée dans laquelle des paquets sont retransmis ultérieurement lorsque les destinations sont indisponibles temporairement, au lieu de les rejeter. Cette extension mène à un taux de remis considérablement plus élevé que OLSRv2 par défaut dans certains scenarios.

Ce rapport présente alors une analyse des menaces de sécurité pour OLSRv2, suivie par la spécification d'un mécanisme de contrôle d'accès pour OLSRv2, qui empêche nombre de ces attaques.

Quatrièmement, ce rapport explore le dimensionnement des réseaux de capteurs sans fil, et évalue un protocole de routage des réseaux de capteurs, nommé RPL, spécifié par l'IETF en 2011. En outre, étant donné que RPL ne contient pas de mécanisme de ≪ broadcast ≫ efficace, plusieurs mécanismes de broadcast optimisés dans RPL sont présentés et comparés.

Acknowledgments

First and foremost, I would like to thank Thomas Clausen, who guided me during my time at LIX, Ecole Polytechnique. I really appreciated his dedication, his vast experience, his teaching approach, his inspiration and his immense pool of stories about air line companies. I greatly enjoyed working in the Hipercom team, in particular with Emmanuel Baccelli and Philippe Jacquet, who were both very helpful in research and administrative matters. My thanks also go to all other (current and former) members of the team: Veronika Maria Bauer, Alberto Camacho Martínez, Ming-Chih Chien, Songyean Cho, Axel Colin de Verdière, Juan Antonio Cordero Fuertes, Marienestor Mariyasagayam, Matthias Philipp, Georgios Rodolakis, Georg Wittenburg, and Jiazi Yi. They made my stay at LIX a great time. Without our charming and hardworking assistants, Lydie Fontaine, Marie-Jeanne Gaffard, Valérie Lecomte, and Cindy Vingadassamy, I would not have survived the administrative part during my time in France.

I greatly enjoyed my teaching experience at Ecole Polytechnique, and would like to thank Julien Cervelle, Christophe Jacquet, Sylvie Putot, Dominique Rossin, Olivier Serre, and Vincent Siles. A special thanks to my friends and colleagues Vivek Nigam, Carlos Olarte, Balaji Raman, and Florian Richoux. I appreciated working together with colleagues from our cryptography team: Andreas Enge, Jérôme Milan, and François Morain.

Aaron Kaplan and Henning Rogge gave me a deep insight in their deployment experience, and I had many intense technical discussions with them, which I enjoyed very much.

I would like to express my gratitude to Ian Taylor, for his kindness, his great programming skills and his collaboration on AgentJ.

A special thanks to all my friends and colleagues that I met at IETF meetings, in particular Brian Adamson, Justin Dean, Yuichi Igarashi, Yannick Lacharité, Charles E. Perkins, Hiroki Satoh, Mark Townsley, Ronald in 't Velt, and Ryuji Wakikawa.

I really appreciate the work of the reviewers of the thesis, Michel Diaz, Joseph Macker, and Kenichi Mase, as well as the examiners of the thesis, Robert G. Cole, Walid Dabbous and Christopher Dearlove.

Lastly, I profoundly thank my parents and my wife, for all their support during the thesis. My wife should be awarded with a Master of Science in Ad Hoc Networking, since she had to listen to all my chitchat about OLSRv2 and ad hoc networking.

Contents

Part I

Introductory Part

Chapter 1

Introduction

The Internet has grown in size by orders of magnitude since its inception in the late 1970s. According to [1], the number of hosts in the Internet has increased from about 200 hosts in 1981 to more than 700 million hosts in 2010, as depicted in figure 1.1. A similar indication of the scaling of the Internet is illustrated in figure 1.2, showing the development over time of the number of active BGP forwarding entries, as observed by [2].

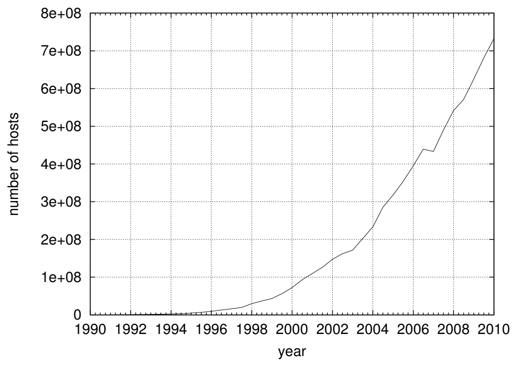


Figure 1.1: Hosts in the Internet (data source: [1])

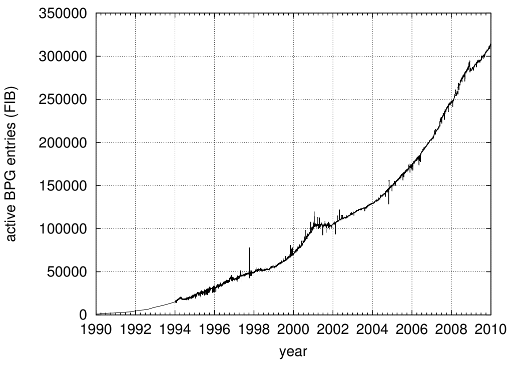


Figure 1.2: Active entries in the BGP Forwarding Table (FIB) (data source: [2])

Despite the growth of the Internet, it has remained roughly unchanged in structure: hosts can communicate with other hosts anywhere in the world, enabled by an intricate system of wires and network infrastructure, called "routers". The structure of the Internet can schematically be divided into two parts: the "*core*" and the "*edge*". The core – typically under administration of Internet Service Providers (ISPs) – contains the required network infrastructure that allows an end-to-end connection of hosts in the "edge" (*e.g.* home PCs and laptops). The reason for separating the network into core and edge is, in part, that hosts on the edge of the Internet need very little knowledge about how to find paths to other hosts, allowing smaller and cheaper devices for the end customers, whereas the more expensive network equipment is hosted by ISPs.

[3, 4] show that the topology of the core is relatively stable, with network components mostly

connected by cables, and the network topology planned by ISPs. With the advent of wireless interfaces in the 1990s, and the resulting mobility of devices, hosts on the edge were enabled to change the “point of attachment” to the core of the Internet. For example, a user with a laptop equipped with a wireless interface, can access the Internet through both the wireless access point at home, and through the employer’s wireless network. Due to the separation of the Internet into core and edge, the mobility of hosts did not require a significant change to the classic model of the Internet, since a mobile host can disconnect from the core, move and reconnect elsewhere without any required change within the topology of the core.

As demonstrated in [5, 6], the Internet *core* does not handle well high topology dynamism and instability, such as faced with router mobility or unstable connectivity between routers due to wireless radio characteristics. As Internet protocols were originally not specifically designed for a dynamic core, new protocols need to be specified which cope with this new kind of network [7].

In the following, the classic network model with a stable core and the new mobile model, in which the core is much more dynamic, are presented.

1.1 Classic Networks

The (simplified) classic model of the Internet is depicted in figure 1.3. The inner gray cloud represents the core of the Internet, whereas the outer white cloud represents the edge of the Internet. The edge is comprised of *hosts*, running applications such as web browsers, text processors or games. Through a network connection via the core, these hosts can run distributed applications, *i.e.* applications that are shared among several hosts. An example distributed application is a web browser, requesting the content of a web site from another host, the web server hosting the requested web site. For both hosts (web browser and web server), the internal structure of the network core is invisible (a “black box”), so they only need to send the data to their “point of attachment” to the core, and are unaware of the path the data takes towards the destination.

The core of the Internet contains *routers* that provide the end-to-end connection for the hosts on the edge of the Internet. A router is, basically, a device with one or several network interfaces, which receives incoming data in form of packets, not destined to the router itself, from one of these interfaces and then selects one of its outgoing interfaces to forward the data. To be able to make the decision to which “next hop” along the path the packet should be sent, the router needs information about the path through the core towards the destination of the data. This information is usually gathered by running a “routing protocol” on the router which, by means of message exchange with other routers, “learns” the topology of the network.

A condition for routers to be able to forward data correctly from the source to the destination is that the routing protocol can cope with changes to the network topology, in which it finds itself, quickly enough to provide an accurate path when the router is supposed to forward data. In the classic network model, the topology of the routers on the core typically changes infrequently [3, 4]: the topology is planned and administrated by system operators of ISPs. Moreover, as routers are mostly connected via cables, their relative topological “position” to other routers rarely changes, and the links between routers are stable.

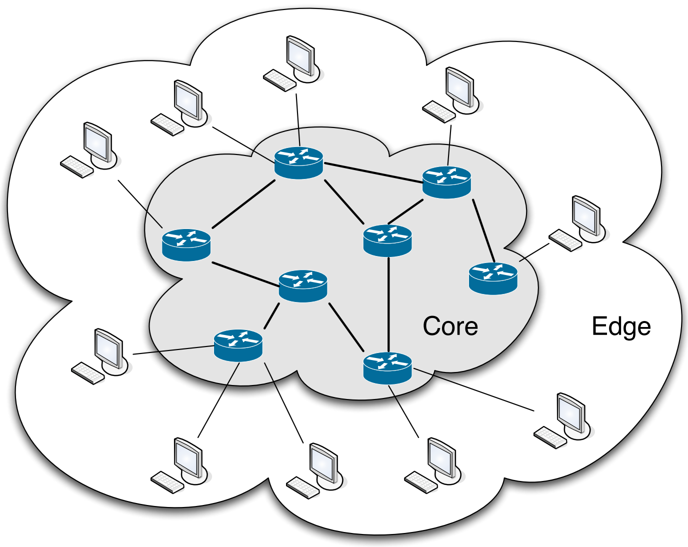


Figure 1.3: Classic Internet model: routers on the core of the network connect hosts on the edge of the network.

A precondition for successful routing is that all entities in the network (hosts and routers) have acquired unique identifiers. These identifiers (IP addresses) are used by hosts to address the destination of the transmitted data, and by routers to calculate paths to the destination and to correctly forward the data throughout the network. In the Internet, a pool of addresses is assigned to each ISP from a global registry of IP addresses, as depicted in figure 1.4[1]. Each ISP assigns IP addresses from its pool to its routers. Customers then get a pool of addresses for their hosts. The hosts can be automatically configured with addresses from the pool. The ISP must have a precise plan of the network topology and the allocated IP addresses of its routers, before it can apply for addresses from the registry [9]. Due to this planned configuration of routers, address conflicts (*i.e.* duplicate addresses of two routers) are rare and can be fixed manually.

Another consideration is that hosts on the edge rely on the correct behavior of routers on the core; if a router does not behave according to the specification (either because it is misconfigured, damaged or compromised), the stability of the network may be reduced and data sent through the network may not reach the correct destination. In the Internet, the risk of such instability due to a misbehaving router is contained by the physical access using a wire to routers on the core; an attacker needs to connect to the core by plugging a wire into a router, which is usually secured in protected server rooms with restricted access.

In the classic model of the Internet, routers on the core are devices, especially designed for their specific tasks: (1) to route large amounts of data per second through the network, to (2) efficiently calculate paths to destinations and (3) to store possibly huge tables of destination IP addresses in memory. Commercial routers are therefore equipped with large storage capacity as

[1]To be accurate, each ISP acquires the pool from a regional address registry (RIR), which in turn acquires its address pools from the Internet Assigned Numbers Authority (IANA). Refer to [8] for a complete description.

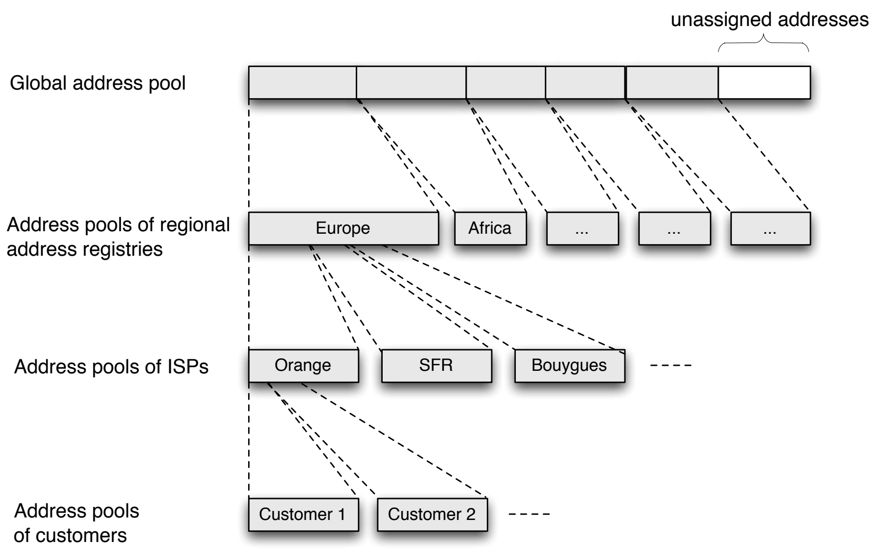


Figure 1.4: Address pools: end customers acquire an address or a pool of addresses from a global registry (through their ISP and a regional registry per continent).

well as powerful processors, contrary to hosts on the edge, which may include hardware with only little CPU power and memory (such as mobile phones or PDAs).

1.2 Highly Dynamic Core

A new class of networks has emerged since the late 1990s [10], which add an additional layer to the static core of the Internet as summarized in section 1.1. The enabling technology for this kind of networks is wireless interfaces, which allow routers in the core of the network to be mobile, creating dynamic topologies, as depicted in figure 1.5 (in the cloud denoted "Dynamic core").

Routers with wireless interfaces form a new kind of network, communicating among themselves over wireless links and thereby forming a dynamic, arbitrary and often multi-hop graph. The core properties of these networks are their autonomy, *i.e.* that they can be deployed with no a priori configuration, and their ability to cope with a high degree of network topology dynamism. Contrary to the classic network model as presented in section 1.1, routers are not necessarily managed and administered by ISPs, but become a commodity, available on every computer or PDA equipped with a wireless interface. In the following, the properties of this new kind of network are presented in more detail. As these new networks form in a spontaneous, ad-hoc fashion, without any preplanning and a dynamic topology, they are denoted "*ad hoc networks*" for the remainder of this manuscript.

Ad hoc networks are often also called "Mesh networks" or "Mobile Ad Hoc Networks (MANETs)", with the connotation that, contrary to a MANET, routers in a Mesh network are not mobile. But since a dynamic topology can be caused by mobility as well as by unstable links due to

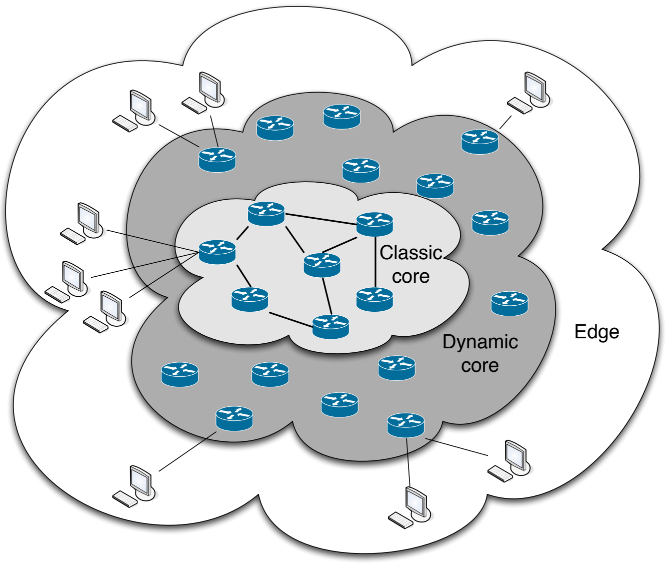


Figure 1.5: New Internet model: mobile routers form a spontaneous, unplanned network with a dynamic topology

wireless radio characteristics, both MANETs and Mesh networks have the same properties, only influenced by the degree of the topology dynamism.

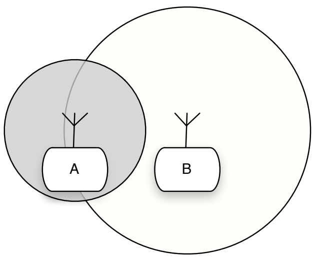
1.2.1 Properties of Ad Hoc Networks

Ad hoc networks have the following main attributes:

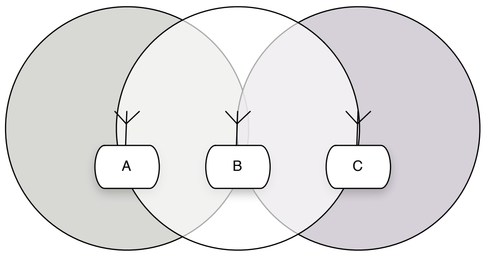
- *Dynamic topology*: A particularity of ad hoc networks is the dynamic topology of the network. Links between routers may change dynamically – due to router mobility, or other factors influencing link stability such as environmental changes, obstacles, etc. – and with a higher frequency than in classic, wired networks. At a given point in time, depending on the mobile routers' positions and stability of the wireless link between two adjacent routers, a wireless connectivity in the form of a random, multi-hop graph or "ad hoc" network exists between the routers [10].

- *Wireless radio characteristics*: In addition to the dynamism of the topology, the following properties of the wireless radio medium affect the topology of the network: *asymmetry* and *non-transitivity* of links. Asymmetry means that, unlike on a wired link (*i.e.* a cable), the fact that a router A "hears" a router B does not necessarily imply the inverse, as depicted in figure 1.6(a).

  Furthermore, transitivity is not guaranteed in ad hoc networks. In figure 1.6(b), three routers A, B and C are depicted, each of them equipped with a wireless interface. Non-transitivity means that router A cannot "hear" C, even though A can hear B, and B can hear C.



(a) Asymmetry



(b) Non-transitivity

Figure 1.6: Wireless radio link properties (A, B, C are routers with a wireless interface each, and the circles around the routers represent the radio range)

- *Infrastructureless*: Typically, no centralized infrastructure exists in ad hoc networks, such as access points in classic networks. Even if a “central” router exists in an ad hoc network, communication between that central router and each other router in the network cannot be guaranteed due to the dynamic topology.
- *Autonomy*: Ad hoc networks are “autonomous”, *i.e.* routers do not need any a priori configuration. All parameters required for establishing communication, such as for the routing protocol, are negotiated between the routers by a message exchange [10].

1.3 Use Cases

In this section, some example use cases of ad hoc networks are presented, that demonstrate the interest and the advantages of ad hoc networks over classic, wired networks in these deployments.

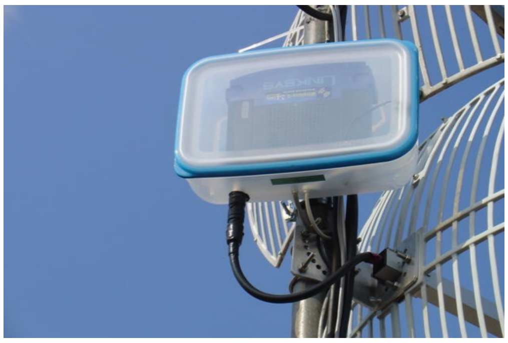
1.3.1 FunkFeuer Community Ad Hoc Network

One example of operational ad hoc networks are community networks in Europe, such as the FunkFeuer [11] network in Vienna, Austria. FunkFeuer is a public, non-regulated, community network allowing inhabitants of Vienna to connect to each other and – through an uplink to an ISP – to the Internet for no fee, other than the initial purchase of a wireless router. Funk-Feuer comprises several hundred routers [11], many of which have several radio interfaces (with omnidirectional and some directed antennas)[2].

The routers of the network are small-sized wireless routers, such as the Linksys WRT54GL, available in 2011 for less than 50 Euros. These routers, with 16 MB of RAM and 264 MHz of CPU power, are mounted on the rooftops of Vienna, as depicted in figure 1.7.

When new users want to connect to the network, they have to acquire a wireless router, install the appropriate firmware and routing protocol, and mount the router on the rooftop. IP address(es) for the router are assigned manually from a list of addresses.

[2]As of August 2010, 749 routers were registered, but only 222 of them were active. The topology contained 1336 links between the active routers.


(a) Initially, simple Tupperware has been used to protect the routers

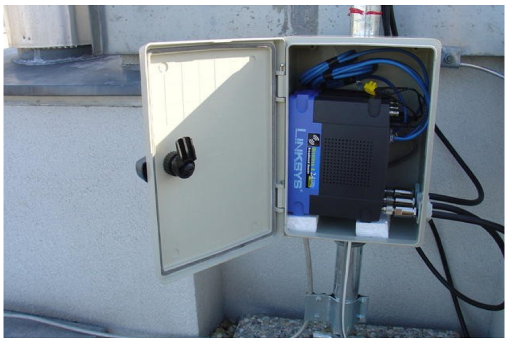

(b) Recently, better protection boxes have been used

Figure 1.7: FunkFeuer routers (source: [11])

This network is *not* intended for testing purposes, and there is no “operator” or “maintainer” having remote access to the routers once installed, which requires the routers to be completely autonomous. Figure 1.8 depicts the network topology of the FunkFeuer network as of July 23, 2010. The network covers a large area of central Vienna, and via some directed links, also some suburbs further away from the city center. While the routers are non-mobile, fluctuations in link quality require an ad hoc routing protocol that allows for quick convergence to reflect the effective topology of the network.

Similar community network deployments are situated, *e.g.*, in Berlin [12], Athens [13] and Seattle [14].

As of 2011, the routing protocol that operates on FunkFeuer routers is unsecured. Therefore, the users of the network have to trust that nobody harms the network integrity (intentionally or otherwise).

1.3.2 Ad Hoc Networks in Emergency Situations

Establishing basic communication after an emergency such as a flood, earthquake or nuclear accident, is difficult when the communication infrastructure is damaged. Mobile phones require nearby infrastructure that provide connectivity, which may not work any more. Even if the infrastructure is still available, the increased use of mobile phones after an emergency can saturate the network. The cable telephone network may be malfunctioning when cables are broken, satellite phones are rarely available and expensive.

In addition to voice communication, data collection on the emergency site is desirable. Information, such as temperature, humidity or radioactivity of the disaster area, can help understanding the degree of the disaster, and to coordinate help accordingly. [15, 16, 17, 18] describe different such deployments, establishing communication in emergency situations. To pick one example, the SKYMESH project of Niigata University [15], is aimed at establishing communication between several unmanned balloons (as depicted in figure 1.9) in order to rapidly create communication networks for rescuers. A small computer, together with a GPS device and a

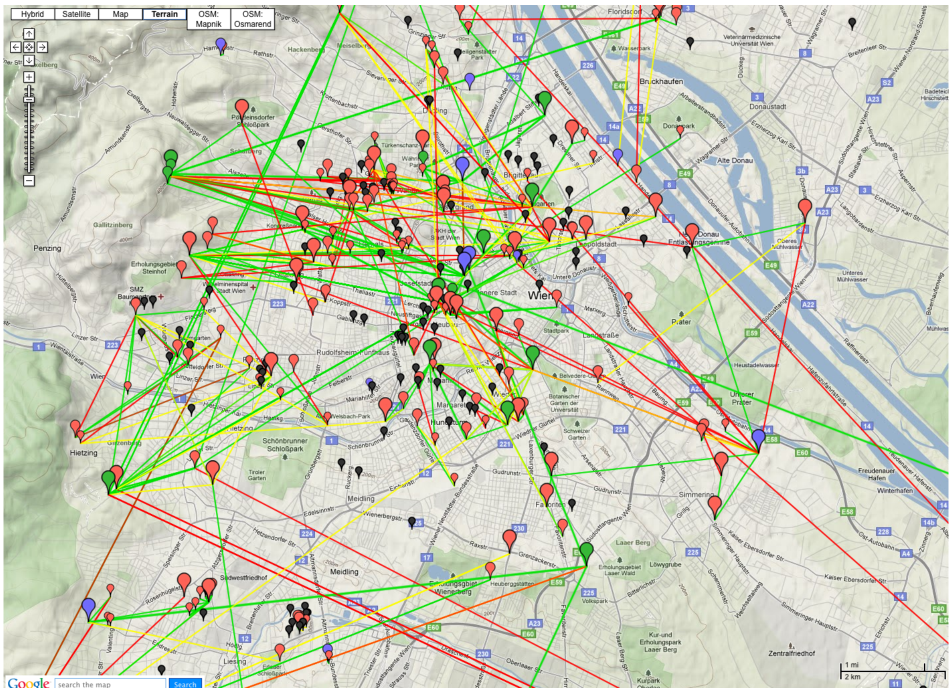
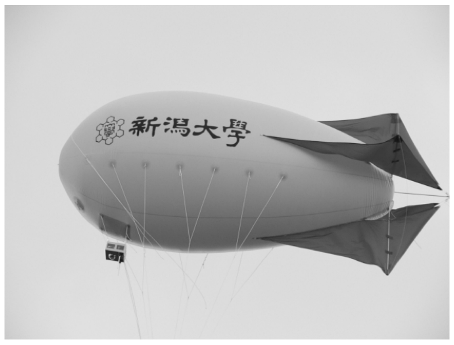
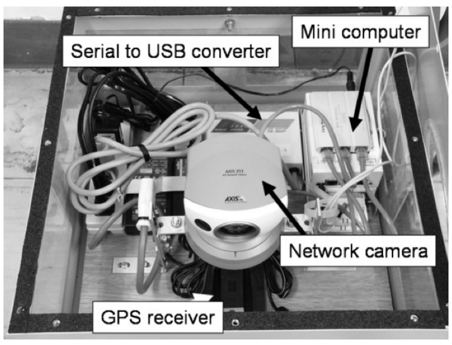


Figure 1.8: FunkFeuer topology (as of July 23, 2010)



(a) Balloons used for establishing an ad hoc network in emergency situations



(b) GPS, camera and computer are attached to the balloon

Figure 1.9: SKYMESH project (source: [15])

camera, is attached to the balloon, which floats in a height of 50 to 100m over ground, allowing remote wide area monitoring of the disaster area, as well as establishing communication (voice or data) using the ad hoc network.

Another deployment in emergency situations is to drop large numbers of sensors from an airplane. The sensors can then establish an ad hoc network, once they are on the ground, without the necessity for humans to enter the disaster site and to deploy the sensors manually.

1.3.3 Wireless Sensor Networks

The general context for Wireless Sensor Networks (WSNs) is small, cheap devices whose primary function is data acquisition, with communications capabilities enabling them to send data to a controller, using a wireless multi-hop topology. As an example, a WSN deployed for environmental monitoring might contain a set of temperature sensors, sending "notifications" to a central controller when the temperature exceeds certain thresholds. Compared to a network of wired sensors, WSNs offer the advantage of enabling mobility to sensors, as well as reducing cost and space requirements for the installation of cables. The properties of WSNs are similar to the ad hoc network presented in section 1.3.1, with the following differences: (1) hardware resources (in terms of CPU and memory) of sensor routers are even more constrained than ad hoc routers in the FunkFeuer network, (2) unlike the routers in the FunkFeuer network, sensor routers may be battery driven, and (3) sensor network topologies are often optimized for particular traffic patterns.

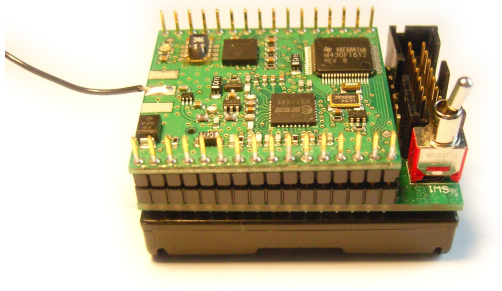


Figure 1.10: Sensor router MSB430 with 55 KByte of flash and 5KByte of RAM (source: [19])

As for (1), figure 1.10 depicts a typical sensor router, which is equipped with only 55 KByte of flash, 5 KByte of RAM, and a few Megahertz of CPU speed [19]. Compared to the routers in the FunkFeuer network, these sensor routers have much more constrained resources [20], and thus require special care when designing protocols for these sensor routers, minimizing required memory space and CPU power. As for (2), sensor nodes, such as [19], are battery-driven, constraining their life-time compared to routers with a permanent energy supply. This implies that protocols for such sensors should have the objective to maximize resource savings (*e.g.* by reducing the frequency of message transmissions). As for (3), a major use case for sensors is measuring a set of environmental data and sending it through the network to a central controller [20]. This traffic flow assumption allows to construct specific, optimized network topologies which focus on connections from a sensor to the controller (versus sensor-to-sensor or controller-to-sensor), such as the topology depicted in figure 1.11.

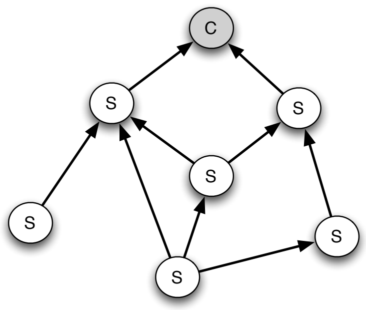


Figure 1.11: Sample wireless multi-hop topology of a Wireless Sensor Network with sensors (S) sending traffic to a controller (C)

1.4 Challenges of Ad Hoc Networks

Section 1.3 has shown use cases of ad hoc networks that demonstrate advantages of this new kind of networks. Before ad hoc networks can be commercially deployed, however, several technical challenges have to be solved, which are summarized in this section (without claiming completeness):

1. As ad hoc networks are designed to be autonomous, and to have an un-planned, possibly dynamic network topology, manual allocation of IP addresses on routers may not be feasible.
2. Routing protocols have to cope with the high dynamism of the topology as well as properties of the wireless radio channel.
3. Router parameters should be remotely manageable in order to change settings on the router and to tweak the network performance.
4. Due to the wireless radio interfaces, physical access to routers in an ad hoc networks is easier than in wired networks, and therefore attacks on the network integrity are facilitated.
5. Ad hoc networks should be integrated in the wider Internet.
6. Due to constrained hardware and bandwidth, ad hoc networks are more difficult to scale.

Autoconfiguration: As described in section 1.2.1, ad hoc networks are (1) autonomous, (2) infrastructureless, and (3) have no preplanned topology with a possibly dynamic network topology. Due to their autonomy, no "a priori" network configuration is required for network deployments, and routers therefore should provide means to configure addresses automatically (called "autoconfiguration"). As for (2), no centralized infrastructure, such as an address registry, is available from which unique addresses can be allocated. As the network topology is not planned, guaranteeing unique and topologically correct addresses at all times is difficult; parts of the network can split and merge at a later point of time, and the relative position between routers may change due to the dynamic topology. Protocols for autoconfiguration in the Internet ([21, 22, 23]) exist, but are not applicable for ad hoc networks, as they have not been designed to cope with the above-mentioned properties of ad hoc networks [24]. Moreover, the autoconfiguration protocols in the Internet are specified for *hosts*, not *routers*, *i.e.* existing protocols have to be extended or new protocols have to be specified for ad hoc routers.

Routing Protocol Challenges: Due to the dynamic and spontaneously forming topology of ad hoc networks, routing protocols have to react to the changing topology of the network and find paths through the network. Current routing protocols in the Internet, such as OSPF [25] and BGP [26], were not designed to handle such dynamic topologies [5, 6] and produce a high message overhead in ad hoc networks, prohibiting their use in these networks [27, 28]. In OSPF, routers synchronize their databases that reflect the topology of the network, which creates a considerable overhead in ad hoc networks, if links between adjacent routers are interrupted due to the dynamic topology [28]. OSPF also specifies to use reliable dissemination of routing protocol messages throughout the network, *i.e.* the protocol verifies that messages are received by other routers using an acknowledgment mechanism; in a wireless medium, this may lead to loss of the acknowledgment messages if they are transmitted at the same time on the channel.

Router Management: Management of router parameters (*e.g.* parameters of the routing protocol, such as message transmission intervals), similarly to address configuration, is more difficult than in classic networks: there may not be any network operators for the network, and even if there are, they might not have access to the router to be managed (either because it is not "owned" by the network operator and protected against external access, or because it is currently in another, separated, part of the network and thus not reachable from the operator's computer).

Security: Security is another major concern for ad hoc networks. While the threats to the operating system, routing protocols and other applications are the same for ad hoc networks as for their counterparts in wired networks, the physical access to the routers is easier in ad hoc networks. It suffices for an attacker to use a computer with a wireless interface in radio range of any router in the ad hoc network in order to attack the integrity of the network, *e.g.* by transmitting control messages reflecting a "wrong" network topology. In wired networks, physical access to the routers is more difficult, since they are typically protected in secured server rooms, and since a connection via cable is required in order to attack the router or the network integrity[3]

Integration in the Internet: It is desirable to integrate ad hoc networks into the larger Internet, in order to allow communication between hosts in the ad hoc network and other hosts in

[3]However, attacks on the BGP protocol – which relies on trust between routers – indicate that even wired networks are not unaffected by security threats.

the Internet. This requires care when designing protocols that should interoperate with current protocols and architectural assumptions in the Internet. The different properties of ad hoc networks, summarized in section 1.2.1, should be transparent (*i.e.* indifferent) to all hosts on the edge and all classic routers (not running an ad hoc routing protocol) on the core, in order to avoid rewriting software for millions of devices already deployed.

Scalability: Increasing the size of ad hoc networks, *e.g.* in order to allow city-wide or even larger ad hoc networks, is challenging because, unlike typical classic routers, ad hoc routers may have constrained hardware power, running on laptops, PDAs or even more constrained devices. Moreover, the bandwidth of the wireless radio medium may be more limited than in wired networks. Routing protocols demand both in-router resources (CPU, memory) and bandwidth for message exchange between routers which, depending on the traffic patterns, may scale with the number of routers in the network.

Finding solutions to these challenges is important, as ad hoc networks are not only part of academic research, but are operated in real life deployments, as presented in section 1.3. This manuscript aims at analyzing the problem space in more detail and at proposing solutions, that help to deploy such networks in industry and academia, and to integrate this new kind of network into the classic Internet.

1.5 Standardization in the Internet Engineering Taskforce

The fact that ad hoc networks are not just deployed in academic lab test beds, but in operational industrial and community networks, shows that there is an interest of industry to market these networks. If companies develop proprietary protocols for ad hoc networks, there is a risk of market fragmentation and non-interoperable protocols [29]. In order to standardize protocols and to avoid such a market fragmentation, the Internet Engineering Taskforce (IETF) [30] publishes specifications of protocols for the Internet. In particular, the IETF has formed several Working Groups (WG) and assigned them with the task of standardizing protocols for ad hoc networks. This section briefly overviews the organization and the standardization process of the IETF. For a complete, detailed description of the structure and processes of the IETF, refer to [31].

According to [32], "the mission of the IETF is to produce high quality, relevant technical and engineering documents that influence the way people design, use, and manage the Internet in such a way as to make the Internet work better". Contrary to other standardization bodies, the IETF is loosely organized: there is no membership, no membership fees, no board of directors, and the IETF is no corporation. Three times a year, interested people from industry and academia meet face-to-face in order to make progress on standardizing protocols. In between these meetings, communication is mainly performed via mailing lists. The opinion of everyone interested in the topic and producing constructive comments is welcome, independent of his or her affiliation with a company or the hierarchy within the company. One unofficial motto of the IETF, expressed by David Clark, is: “We reject kings, presidents and voting. We believe in rough consensus and running code” [31]. That means that (1) decisions on controversial questions are not made by hierarchical decisions or votes, but by *rough* consensus on the mailing list (*i.e.* not necessarily unanimous, but no major technical objection) and (2) that protocols must also be implemented and tested before they can be standardized.

The Working Groups in the IETF are organized in different areas, as depicted in figure 1.12 (with only Working Groups listed that work on ad hoc network protocols which are considered in this manuscript, notably MANET, Autoconf and ROLL). Each area is guided by two area

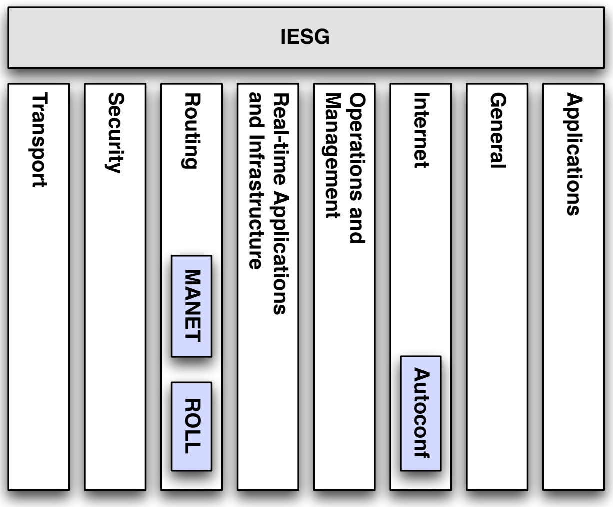


Figure 1.12: Organization of Working Groups in the IETF: Working Groups (here: MANET, ROLL, Autoconf) are organized in different areas

directors, which are part of the Internet Engineering Steering Group (IESG). Nominated for a duration of two years, the ADs administer the publication process according to a set of rules and procedures [31]. The IESG ratifies or corrects the output from the IETF’s Working Groups [31].

Before a document is published as “Request for Comment (RFC)”[4], it has to run through a standardization process, typically starting as individual draft, which can be submitted by everyone without peer review. As such, it is available for discussion by other interested participants on the mailing list, but simply expresses an individual proposal of a person. If a Working Group is interested in the work, it can agree (by rough consensus) that this draft becomes a Working Group draft. Once the Working Group agrees that after a certain number of revisions of the draft, it is mature for publication, it will issue a Working Group last call. During this limited time period, substantial issues and comments should be expressed by members of the Working Group. The draft will then be sent to the IESG, which determines whether the draft is ready to be published, and in particular does not conflict with other, already published RFCs.

[4]Contrary to what the name “Request for Comment” implies, RFCs are the standards specified by the IETF in their final version.

Several of the Working Groups of the IETF focus on protocols that apply to ad hoc networks. For this manuscript, in particular the work of the MANET, AUTOCONF and ROLL Working Group is considered.

1.5.1 MANET Working Group

Created in 1997, the MANET Working Group aims at developing routing solutions in the field of mobile ad-hoc networks [7]. The purpose of the MANET Working Group is

> "(...) to standardize IP routing protocol functionality suitable for wireless routing application within both static and dynamic topologies with increased dynamics due to node motion or other factors." [33]

The Working Group has published the following modular documents that may serve as building blocks for other ad hoc (routing) protocols:

- "*Jitter Considerations in Mobile Ad Hoc Networks (MANETs)*" [34]
- "*Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format*" [35]
- "*Representing Multi-Value Time in Mobile Ad Hoc Networks (MANETs)*" [36]
- "*Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)*" [37]

DYMO [38], OLSRv2 [39] and SMF [40] are protocols which use the above-mentioned modular components, and which are in the process of standardization by the MANET Working Group as of 2011.

1.5.2 AUTOCONF Working Group

The AUTOCONF Working Group charter [41] from 2009/2010 states[5]:

> "In order to communicate among themselves, ad hoc nodes [...] need to configure their network interface(s) with local addresses that are valid within an ad hoc network. Ad hoc nodes may also need to configure globally routable addresses, in order to communicate with devices on the Internet. From the IP layer perspective, an ad hoc network presents itself as a L3 multi-hop network formed over a collection of links."

[41], as of 2009/2010, furthermore states the objective: to "describe the addressing model for ad hoc networks and how nodes in these networks configure their addresses". This manuscript discusses the issues that the AUTOCONF Working Group faces in chapter 4, and proposes an architectural model that adheres to the architecture of the Internet. Core assumption of the proposed model have been documented in [42].

[5]The Working Group has been rechartered on December 21, 2010

1.5.3 ROLL Working Group

ROLL is an abbreviation of an IETF Working Group named “Routing Over Low power and Lossy networks”. This Working Group has as objective to develop a routing protocol for WSNs, based on IPv6.

The unofficial goal, which this Working Group tries to attain, is to prevent fragmentation in the WSN market by providing an IP-based routing standard and solicit broad industrial support behind that standard. As of 2011, the ROLL Working Group is mainly working on the “Routing Protocol for Low Power and Lossy Networks” (RPL)” [43], which is described and further evaluated and extended in this manuscript.

1.6 Contributions and Structure of the Manuscript

This manuscript is structured into four main parts (plus a conclusive part and appendices), each focusing on solutions to a different problem area, as described in section 1.4.

Part I contains, in addition to this introduction chapter, two chapters describing developed software tools which are applied throughout this manuscript:

Chapter 2 first presents a preexisting software tool, called NS2, which is used for protocol evaluations using network simulations throughout this manuscript. Using a simulator allows to understand and evaluate high-level properties of routing protocols and extensions. Simulations are often easier to perform than building a large testbed network of routers, simulating mobility, and guaranteeing the reproducibility of predefined scenarios. Then, the chapter details several tools that have been developed, which allow to automate the process of defining scenarios for the simulator, creating the scenario files required for NS2, running the simulator using these files, evaluating the output of the simulator and plotting figures, all in a single step.

Chapter 3 details how to run unmodified Java applications in the C++ based NS2 network simulator, *i.e.* using the very same Java code that runs on a real network, using a tool called AgentJ. In the remainder of the manuscript, several protocols have been implemented in Java, due to the platform independence, ease of prototyping and extensibility. An extension to AgentJ is proposed which allows for also testing *routing protocols* in NS2. The necessary architectural changes to AgentJ are described in the chapter. When using a routing protocol in NS2 with AgentJ, only a single extra Java class has to be added to the routing protocol, and NS2 does not have to be recompiled for new routing protocols, once AgentJ is installed. The simulation of a basic Java routing protocol is compared with a similar routing protocol implemented in C++, showing that the memory and CPU requirements for the simulation with AgentJ compared to the C++ protocol are only moderately higher. These contributions have been published in [44].

Part II proposes an architectural model that allows to integrate ad hoc networks into the wider Internet. Moreover, this model facilitates scalability of ad hoc networks by enabling automatic configuration of IP addresses on network interfaces and delegating prefixes to ad hoc routers, which is advantageous over manual configuration in large networks:

Chapter 4 contains an architectural model for allowing ad hoc networks to be integrated seamlessly into the Internet. The model compares the current architecture of the Internet with a commonly assumed model of ad hoc networks, and shows that this ad hoc model is not compliant with the Internet. The chapter proposes an alternative model, which avoids these problems and sets the stage for new autoconfiguration algorithms adhering to that model. These contributions have been published in [45, 24] and as contribution to RFC5889 [42].

Chapter 5 specifies an autoconfiguration algorithm, and extensions thereof, adhering to the model presented in chapter 4. The proposed protocol is formally verified using a model checker, which proves correctness of the algorithm. The performance of the algorithm is tested using network simulations in scenarios with increasing number of routers, showing that with some optimizations, the algorithm is performant. These contributions have been published in [24].

Part III explores performance improvements, scalability, management and security extensions of the Optimized Link State Routing Protocol version 2 (OLSRv2) [39], an ad hoc routing protocol which is being standardized by the MANET Working Group in the IETF as of 2011.

Chapter 6 provides an overview of the specification of OLSRv2.

Chapter 7 describes a Java implementation of OLSRv2 (called "JOLSRv2"), which is used for performance analysis and evaluation of extensions in the following chapters. The implementation contains all specified features of the routing protocol, has been shown to be interoperable with other implementations, and can be easily used for prototyping of extensions. These contributions have been published in [46].

Chapter 8 proposes an optimization of OLSRv2 for more efficiently calculating shortest paths from a router to all other destinations in the network. Typically, OLSRv2 runs a variation of the well-known Dijkstra algorithm for path calculation. This implies that for each incoming control message with new topology information, the previously calculated paths are deleted and a new graph is calculated on the router. This chapter proposes the use of a dynamic (*i.e.* incremental) shortest path calculation algorithm (DSP) in OLSRv2, and presents a performance evaluation of OLSRv2 with such a DSP, showing that the performance of OLSRv2 can be increased by orders of magnitude without changing compatibility with the OLSRv2 specification. Using the proposed extensions, topologies of tens of thousands of routers can be handled by standard hardware components, as available in 2011. These contributions have been published in [47].

Chapter 9 presents means for externally managing and monitoring OLSRv2 with the Simple Network Management Protocol (SNMP), a prevalent network management protocol in the Internet, in order to optimize parameters of the routing protocol. Such an optimization may lead to a more stable perceived topology and to a lower control traffic overhead, and therefore to a higher delivery success ratio of data packets, a lower end-to-end delay, and less unnecessary bandwidth and energy usage. The chapter proposes a management framework for OLSRv2, including management objects for controlling OLSRv2 and gathering performance related statistics of the protocol. Moreover, the performance of SNMP in OLSRv2-routed MANETs is tested in network simulations. These contributions have been published in [48] and in IETF working group drafts [49, 50], which are on track to become RFCs in 2011.

Chapter 10 proposes a "delay-tolerance" mechanism to be used in OLSRv2-routed MANETs. Typically, successful packet forwarding on a router depends on whether the routing table of the operating system contains a "valid route" towards the destination of the packet. If such a valid route exists, the packet is retransmitted, otherwise, it is dropped. In ad hoc networks, that behavior may lead to low delivery ratio of packets if the network topology is "unstable". In some deployments, it may be preferable to accept a higher delay of packet delivery by retransmitting the data later when a valid route exists again to the destination, instead of dropping it. This chapter proposes such a delay tolerant mechanism for OLSRv2-routed MANETs, and evaluates the performance by means of network simulations in increasingly large MANET deployments, showing a large increase in delivery ratio at expense of higher end-to-end delays. In order to test the effect of delay tolerant routing in "extreme" cases, a special network model for simulations is proposed, and the mechanism tested in such scenarios.

Chapter 11 sets the stage for a security extension of OLSRv2 by analyzing security vulnerabilities and inherent resilience of OLSRv2. Attacks are classified in three types: (i) attacks that affect the ability for any given router in the network to acquire a representation of the network connectivity, (ii) attacks that have impact on the ability of a router to acquire an accurate topology map which reflects the effective topology map, and (iii) attacks that lead to an inconsistent topology among several routers in the network. For each of the three classes, several attacks are exemplarily demonstrated, and finally, inherent resilience of OLSRv2 (without security extension) to some of these attacks is described. These contributions have been published in [51, 52].

Chapter 12 proposes a security extension to OLSRv2 and evaluates the scalability of that extension. Many of the attacks presented in chapter 11 are only possible because OLSRv2 routers per default "trust" each other. This chapter specifies a security extension to OLSRv2 allowing to control access of routers to the network, and to verify "links" that each such admitted router advertises in its control messages. A simulation evaluation compares the CPU time for processing and generating control traffic signatures, as well as the message overhead, for both the "link-admittance" and the "router-admittance" mechanism. These contributions have been published in [53, 54] and in a IETF Working Group draft [55], which is on track to become RFC in 2011.

Part IV explores routing performance and scalability issues in Wireless Sensor Networks (WSNs):

Chapter 13 provides an overview of the specification of the IPv6 Routing Protocol for Low Power and Lossy Links (RPL), a routing protocol specially targeted for large networks of constrained sensors.

Chapter 14 presents a Java implementation of the RPL protocol (called "JRPL"), which is used for performance analysis and evaluation of extensions in the following chapters.

Chapter 15 evaluates the performance and scalability of RPL in network simulations up to thousand routers, and tests the impact of the protocol on unicast data performance, showing that RPL has a very low overhead on the network and facilitates successful unicast data traffic communication for large sensor networks. These contributions have been published in [56].

Chapter 16: As RPL does not contain any mechanism for flooding information throughout an RPL-routed WSN, several such flooding mechanisms are proposed and compared in their performance with increasing number of routers, showing that optimized flooding mechanisms can largely increase the efficiency in terms of reduced message overhead and number of packet collisions. These contributions have been published in [57, 58].

Part V summarizes and concludes this manuscript, and outlines some possible future work.

Part VI contains a list of publications, as well as more detailed usage and installation instructions for AgentJ, JOLSRv2, and JRPL.

Chapter 2

Simulation Toolchain

An important step when designing and developing a new protocol is often to verify its behavior in a network simulator, such as NS2 [59]. While network simulators have their limits, especially in terms of the fidelity of the lower layers and, for wireless network interfaces, the fidelity of the propagation model used for representing the behavior of radio waves, their use facilitates the understanding of high-level and algorithmic properties of a given routing protocol. In particular, in the area of ad hoc networks, simulations are often easier to perform than building a large testbed network of routers, simulating mobility, and guaranteeing the reproducibility of predefined scenarios. Using NS2, scenarios can be defined to simulate not only the network traffic but also the underlying mobility patterns that affect signal strengths between routers as the scenario progresses.

This chapter briefly overviews the functionality of the NS2 simulator and then describes several tools that have been developed for the purpose of running unsupervised simulations of protocol implementations and extensions in NS2. Automating the process of network simulations is useful for simulating protocols and extensions with a large parameter space and many different scenarios. The tools allow for running many simulations without (much) human interaction. They also facilitate to perform statistics over all scenarios and create plots as images. In the following sections, the different tools are presented, and then a typical workflow of how to run a simulation and perform an evaluation of parameters is described.

It should be noted that there exist network simulators other than NS2, such as QualNet [60], OMNeT++ [61], and NS3 [62]. The reason for using NS2 throughout the evaluations in this manuscript is threefold: (i) NS2 is open-source, which allows to modify the code base, if required, and to save licensing fees, (ii) NS2 contains a variety of standard Internet protocols and link layer types, which allows to consider all aspects a protocol in a simulation, and (iii) NS2 is widely used in research, which allows to compare results and which may increase confidence in the correctness of results (as described in section 3.2). NS3, the successor of NS2, was not used for the evaluations presented in this manuscript because it was still under development, and the code base has largely changed in the last few years (more than 2 million lines of source code have changed from 2007 to 2011 [62]).

Since in this manuscript only NS2 has been used, a description of other network simulators is out of scope. [63] presents different network simulators and compares them in terms of performance requirements and scalability.

2.1 Chapter Outline

Section 2.2 overviews the functionality of the NS2 simulator. Section 2.3 describes tools that are used to automatically generate many scenarios from a given set of configuration files. Section 2.4 provides an overview of tools which allow for running NS2 simulations automatically without human interaction. Section 2.5 presents tools that can be used to perform statistics over the simulation results and to create images that can be included in, *e.g.*, scientific papers. Section 2.6 describes the typical workflow of preparing, running and evaluating simulations using all the presented tools. Section 2.7 concludes the chapter.

2.2 The NS2 Simulator

NS2 [59] is a discrete event network simulator, written in C++, intended for simulations of network protocols between several nodes with predefined mobility and traffic patterns. NS2 is single threaded, and events are scheduled to be executed at a certain simulation time using callbacks. NS2 contains implementations of all major protocols for each layer on the IP stack, such as different physical media, MAC layers, routing protocols and implementations of TCP and UDP. Additional protocol implementations for each layer can, if desired, be added to NS2.

The simulation scenarios are defined in OTcl [64], an object oriented scripting language. The scripts define (1) basic scenarios settings (number of nodes, physical channel, MAC type, routing protocol, etc.), (2) mobility settings of nodes, and (3) data traffic patterns.

A complete description of NS2 can be found at [59].

2.3 Scenario Generation Tools

This section presents tools for automatically generating scenario files to be used by network simulators such as NS2. These scenarios files all correspond to the same abstract scenario description (specified by a configuration file).

2.3.1 Wsg – The Scenario Generator

The purpose of the Wsg scenario generator is to create different scenario files that represent complete scenario files, *i.e.* not only router movement patters, but a standalone scenario file including router initialization, data traffic patterns and movements. This eases the distribution of the scenario files on several machines that run network simulations in parallel. It is written in Ruby and is generic for different simulators, *i.e.* it has its own intermediate representation of the scenario and eventually prints out a file that is particular to one network simulator or the other. So far, only an NS2 output generator is implemented.

The `wsg` directory has the following contents:

```
|-- conf
|   `-- JOLSRv2-20-waypoint.conf
|-- create-multiple-runs.sh
|-- generic
|   |-- BulkTransfer.rb
|   |-- Generators.rb
|   |-- Globalparameters.rb
|   |-- LineSegment.rb
|   |-- LinearMovement.rb
|   |-- Movements.rb
|   |-- Node.rb
|   |-- NodeGroup.rb
|   |-- Popup.rb
|   |-- Random_walk.rb
|   |-- Random_waypoint.rb
|   |-- Scenario.rb
|   |-- Singlelane.rb
|   |-- Stream.rb
|   `-- Vector.rb
|-- scenarios
|-- simulators
|   |-- Human.rb
|   |-- Ns2_full.rb
|   |-- Simulators.rb
|   `-- ns2.rb
|-- wsg.conf
`-- wsg.rb
```

2.3.1.1 Config Files

The `conf` directory contains all config files that are used to create the scenario files. Every config file represents a single abstract scenario description. However, several "runs" of every scenario can be created using the same settings but different random initialized router positions and movements. An example of such a configuration file is listed in the following:

```
GLOBAL.chan Channel/WirelessChannel   # channel
GLOBAL.prop Propagation/TwoRayGround  # radio propagation
GLOBAL.netif Phy/WirelessPhy          # physical medium
GLOBAL.mac Mac/802_11                 # MAC layer
GLOBAL.ifq Queue/DropTail/PriQueue    # interface queue
GLOBAL.ll LL                          # link layer
GLOBAL.ant Antenna/OmniAntenna        # antenna type
GLOBAL.ifqlen 50                      # size of interface queue
GLOBAL.simulation_time 300            # time of simulation in seconds
GLOBAL.width 1000                     # width of simulation area
GLOBAL.height 1000                    # height of simulation area
GLOBAL.routing_protocol JOLSRv2       # used routing protocol
GLOBAL.nn 30                          # number of routers
GLOBAL.simulator NS2_full             # output filter for  NS2 scenario files
#GLOBAL.seed 12345                    # seed for RNG (commented out)

cluster D 20 random_waypoint   # 20 routers out of the 30 move according
                                          # to the random waypoint model
D.streams 5.0                 # avg. number of concurrent data streams
D.stream_packetsize 64        # size of each packet in bytes
D.stream_packetinterval 0.02  # packet interval in seconds
D.stream_duration 10          # duration of each stream in seconds
D.stream_starttime 5.0        # stream starts x seconds after simulation begin
D.stream_stoptime 250.0       # the last stream will end no later
D.mobility.router_resttime 0-5  # pause time after walking
D.mobility.router_speed 2-8     # speed in m/s
D.mobility.router_distance 100  # distance of the same direction

cluster C 10 popup     # 10 routers out of the 30 move according
                                            # to the popup model
C.streams 5.0
C.stream_packetsize 64
C.stream_packetinterval 0.05
C.stream_duration 10
C.stream_starttime 5.0
C.stream_stoptime 250.0
C.mobility.router_visibilitytime 20-35 # routers will be visible
                                       # for x sec.
C.mobility.router_disappeartime 20-35  # routers will be invisible
                                       # for x sec.
C.mobility.router_distance 100-200     # routers will reappear
                                       # in a max. distance
```

In Wsg, routers are separated in so called “clusters”. These represent a number of routers that share certain settings. There is always one “super”-cluster that sets properties for all routers. This cluster is called GLOBAL. General parameters such as the number of routers or the routing protocol are defined in the GLOBAL cluster.

In this example, there are two clusters, called C and D whose routers have different mobility patterns. The following mobility patterns are defined: `random_waypoint`, `random_walk` and `popup`.

2.3.1.2 Generic Scenario Representation

The Ruby classes in the `generic` subfolder contain an internal representation of the scenario file including movement patterns (such as popup or random waypoint), traffic generation, random number generators, router properties and general simulation parameters.

2.3.1.3 Scenario Files

In the subfolder `scenarios`, Wsg stores the output generated scenario files. A script, presented in section 2.4, allows to launch NS2 for all scenario files in that folder.

2.3.1.4 Simulator Output Filters

In the subfolder `simulators`, all output filters for the different network simulators, such as NS2, are stored. While Wsg stores all scenario parameters in an internal data representation, NS2 or other simulators expect their own scenario file format. Wsg allows to pretty print its internal representation to any simulator. At the moment, only a complete NS2 filter has been implemented.

2.3.1.5 Launching the Wsg Scenario Generator

The Wsg scenario generator requires a config file piped as input. It can be launched as in the following example:

```
./wsg < conf/my_config_file.conf
```

The ruby script outputs the resulting scenario file on the standard output.

2.3.2 create-multiple-runs.sh

Multiple scenarios and several runs for each scenario can be created with a batch file by running:

```
./create-multiple-runs.sh -r <number-of-runs>
```

This auxiliary script runs the Wsg generator for every config file in the `conf` directory and creates `number-of-runs` runs for each scenario.

2.4 NS2 Tools

The launch-batch-simulations-launcher.sh script launches unsupervised simulations with NS2 using the provided scenario files. It expects one command line argument `<folder>`, specifying the folder containing the scenario descriptions. The script loads all `tcl` files from the `wsg/scenarios/<folder>` subdirectory. The expected naming convention of these scenario files is

```
.*-[[:digit:]]*-.*-[[:digit:]]*.tcl
```

(represented as standard POSIX regular expression), *e.g.* JOLSRv2-20-waypoint-1.tcl. The first digit group represents the number of routers, the second number represents the nth run of the simulation.

The output of each simulation will be saved in a new subdirectory with the name `<folder>`. Note that the script will not overwrite already performed simulations, and therefore allows to continue interrupted simulation runs.

2.5 Evaluation Tools

These tools are used after a complete run of simulations using the tool described in section 2.4. Their purpose is to perform statistics over the simulations, outputting data and figures that can be used in, *e.g.*, a scientific paper.

2.5.1 NS2stats.awk

The NS2stats.awk script performs some basic statistics for a single trace file. In the following, an example call is listed:

```
 ./NS2stats.awk JOLSRv2-10-1.tr
         Traffic:        Sent:          11187 frames
                         Sent:          715968 bytes
                         Received:      4659 frames
                         Received:      309776 bytes
                         Delivery rate: 0.4165



         AODV/OLSR/OLSRv2/DSR:
                        Bytes:          92979
                        Packets:        1006
                        Messages:       1009
                        Overhead:       92.15 bytes/OLSR_message
                        Overhead:       309.96 bytes/sec

         Path Length:
                Avg. delay:             35.8355
                Avg. pathlength:        1.4606
```

2.5.2 plot.rb and result.rb

`plot.rb` is a Ruby script that launches NS2stats.awk (as described in section 2.5.1) for every trace (*.tr) file in the current directory. It calculates average numbers for all runs of a single scenario simulation. It then outputs a table that can be used for GNUplot [65] (refer to section 2.5.3). An example output looks like:

```
[herberg@adrien test]$ ./plot.rb
reading JOLSRv2-10-1.tr
reading JOLSRv2-40-1.tr
reading JOLSRv2-30-1.tr
reading JOLSRv2-50-1.tr
reading JOLSRv2-20-1.tr

# routers        Delivery ratio  Traffic Sent Frames     Traffic Sent
Bytes      Traffic Received Frames Traffic Received Bytes
Control Traffic Bytes
50      0.72  11221       718144      8180        1085540     2782543
40      0.72  11557       739648      8389        1060868     1809648
30      0.55  10473       670272      5836        6415220      797075
20      0.49  12743       815552      6368        412424      242459
10      0.41  11187       715968      4659        309776      92979
```

2.5.3 stats.plot

`stats.plot` is a configuration file for GNUplot. It expects a table created by `plot.rb` (refer to section 2.5.2). It plots each of the columns as a figure (in `eps` format). A sample figure is depicted in figure 2.1 (without any particular meaning of the plots in the figure).

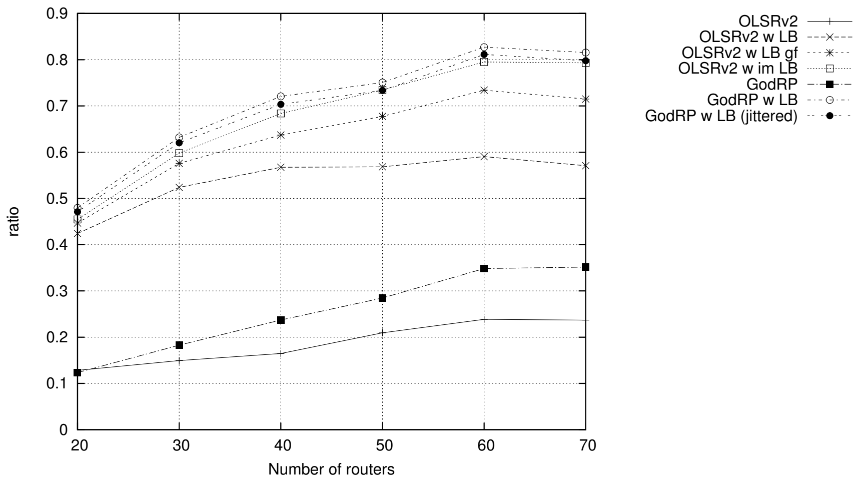


Figure 2.1: Sample output from stats.plot

2.6 Typical Workflow of a Simulation and Evaluation

This section describes a typical workflow of preparing, running and evaluating NS2 simulations using all the presented tools.

2.6.1 Generating the Scenario Files

1. Before creating the simulation scenarios, a set configuration files needs to be created, which represent the abstract description of the scenarios. Refer to section 2.3.1 for a sample configuration file. The config files are saved in the `wsg/config` folder.

2. `./create-multiple-runs.sh -r <any-number>` is executed, where `<any-number>` represents the number of runs (must be higher than 0) for each scenario. This will create the scenario files in the `wsg/scenario` subfolder.

2.6.2 Running the Simulation

1. If the machine intended to run the simulations is accessed remotely (*e.g.* using SSH), it may be useful to execute all scripts in a `GNU screen` [66]. This "virtual" console allows reconnecting or closing the connection while continuing all running processes. This means the SSH connection can be closed, enabling to reconnect later to the machine using SSH, reconnect to the screen with `screen -r` and preserve the same console output.

2. `launch-batch-simulations-launcher.sh <folder>` is executed, launching the unsupervised simulations of all scenario files in the `wsg/scenarios/<folder>` folder.

2.6.3 Performing Statistics

1. `./plot.rb <folder>` is executed, creating a new file called `<folder>.dat` containing the table with the statistics.

2. `gnuplot stats.plot` is executed, creating several `eps` figures.

2.6.4 NS2 Trace File Visualizer

This applet, depicted in figure 2.2, allows for importing NS2 trace files and to display the router topology over time (similar to "nam" or "iNSpect" [67]). Contrary to the latter, no config file has to be written, all parameters – including field size and simulation time – are extracted from the trace file. The applet does not need any installation but runs in a web browser providing a JVM plugin. The applet not only displays the positions of the routers over time, but also draws edges between routers that represent the perceived topology of each router (*i.e.* its neighbors, two-hop neighbors and routers from the topology set). For this to work, the routing protocol must output the local router's view of the topology in the NS2 trace file whenever either of these has changed. An example for such a line in the trace file would be:

```
L 0.63610 14 OLSRv2 N 13 MPR 20 2N 20;28 T ;
```

meaning that at time 0.6361 seconds, router 14 has symmetric links to its neighbors 13 and 20, and that 20 is selected as MPR[1]. Router 14 has 28 as two-hop neighbor through 20, and does not have any tuples in its topology set.

The applet allows for picking one particular router and displaying its radio range as a gray circle around the router, its symmetric one hop neighbors with blue edges, the selected MPRs with red edges, the two-hop neighbors with orange edges, and its topology set with green edges. Thus, the topology of a particular router at a certain time can be easily depicted.

The trace file visualizer allows also to depict the calculated routes of each router, when the trace files contains lines of the following form:

```
L 0.63610 12 OLSRv2 E 12;20;20;14;20;28;20;13
```

Each edge that is part of the routing tree is represented as a couple of two router IDs. In this example, router 12 has chosen the following edges in its Routing Set: an edge from itself to router 20, an edge from 20 to 14, from 20 to 28, and from 20 to 13.

2.7 Conclusion

This chapter has described a number of tools that have been developed for performing unsupervised automatic simulations with JOLSRv2 in NS2 and for performing statistical evaluations of the simulation results. These tools enable performing long simulations with many scenarios or different parameter settings in automated batch-runs without (almost) any human interaction. They also facilitate to run several instances of NS2 in parallel on different computers. The statistical tools allow for creating figures and tables for a whole set of simulations.

[1]Note that MPRs and 2 hop neighbors may be specific to the OLSRv2 [39] routing protocol; however, the logging for other link state routing protocols works without modification: MPR and 2-hop sets are simply left empty

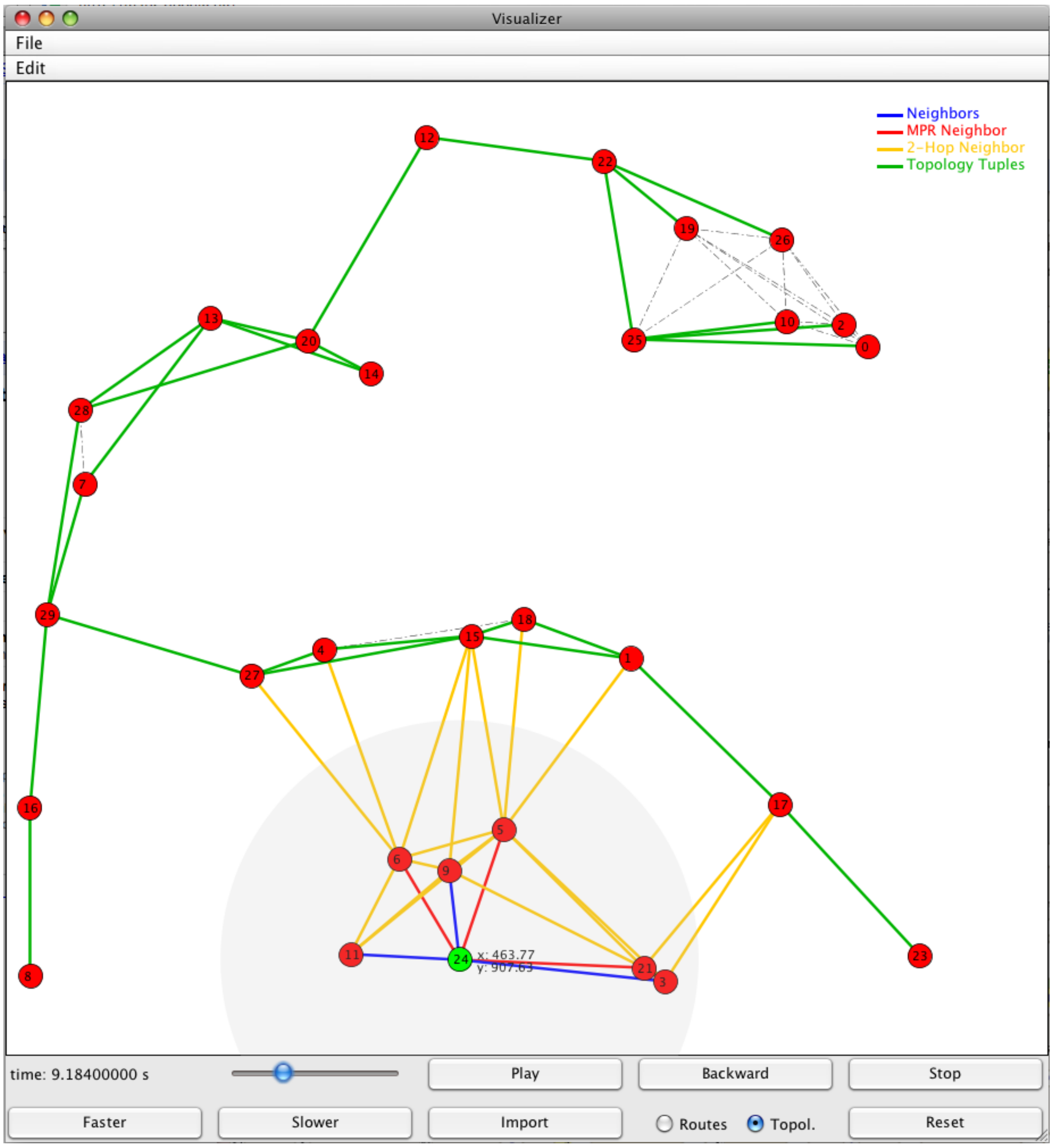
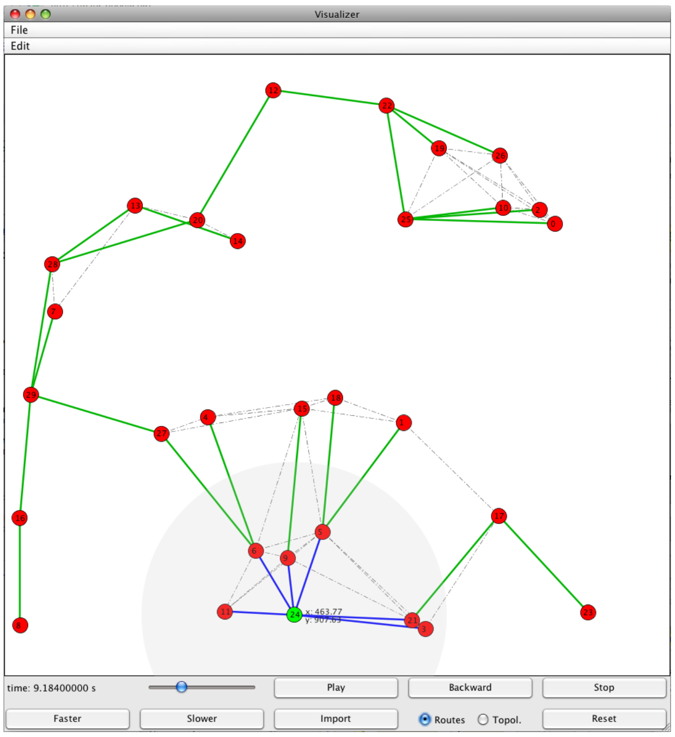


Figure 2.2: NS2 trace file visualizer: displaying neighbor-/2-hop- and topology-tuples of a selected router



time: 9.18400000 s

[✓ Routes] Topol.

Figure 2.3: NS2 trace file visualizer: displaying the routes of a selected router

Chapter 3

Framework for Supporting Java NS2 Routing Protocols (AgentJ)

Routing protocol implementations for NS2 need to be implemented in C++. Furthermore, implementers are required to discretize their application into a sequence of events for simulation rather than running actual code. AgentJ [68], a toolkit developed by the Naval Research Laboratory (NRL), allows Java applications to be run unaltered within NS2. AgentJ is a set of instrumentation modules and classes that collectively integrate Java, C++ and Tcl to enable unmodified Java applications, which currently run on network interfaces and operating systems such as Linux or Mac OS, to be used for an NS2 simulation. AgentJ uses a combination of Java bytecode rewriting and aspect oriented programming techniques, to convert all socket, I/O and timing APIs into low-level C++ methods that bind to the various NS2 counterparts. AgentJ therefore takes an "as-is" Java multi-threaded application and converts it into a thread-synchronized "serial" NS2 version, capable of running within a single-threaded discrete time network simulation environment.

However, AgentJ originally only targeted the application layer agents, and did not consider the encapsulation of NS2 routing agents. This chapter presents the extensions made to AgentJ, which enable it to use Java routing protocols within the NS2 environment. Moreover, this routing architecture is illustrate through the use of specific examples that show how to instantiate and run a new routing protocol implemented in Java, in NS2.

The extensions have been integrated in the AgentJ code base.

3.1 Chapter Outline

Section 3.2 describes related work. Section 3.3 gives an overview of the preexisting tool AgentJ without support for routing protocols. Section 3.4 details what architectural changes have been made to in order to support running a Java routing protocol within NS2. Section 3.5 presents a performance comparison between a Java protocol implementation using AgentJ and a C++

agent. Section 3.6 summarizes the advantages of AgentJ with the routing protocol extension, and section 3.7 concludes this chapter.

In appendix B, a step-by-step instruction is provided how an existing Java routing protocol implementation can be run in NS2, using the extension that is presented in this chapter.

3.2 Related Work

Network simulations of routing protocols or extensions thereof represent an important part of research in the area of MANETs. According to the survey conducted in [69], 75% of all papers published at the MobiHoc conference in the years 2000 to 2005 contain empirical results obtained using network simulations; and further, 44% of which used NS2 simulations. While network simulators have known limitations, they provide a quantitative empirical means of enabling better understanding high-level and algorithmic properties of routing protocols. Due to the importance of analyzing routing protocols in network simulators, it is crucial that the results from the simulation are also adequate for real networks. AgentJ, in this respect, represents a clear novelty in its construction, because it allows the simulation of the Java implementation of the routing protocol, which is the exact same code that will be deployed onto a real network. Therefore, AgentJ allows to test actual working code, rather than discrete approximations that are converted into a non-threaded discrete network simulation environment in order to meet its specific format.

AgentJ builds on a tool named Protolib [70], which in some respects has similar properties to AgentJ but which targets C++ applications. Protolib provides an abstraction layer for C++ classes representing TCP, UDP and Multicast sockets, timers and IP addresses in an abstract and reusable fashion. By rewriting the C++ application to use the Protolib C++ API, and by setting compiler directives before compilation, the same application can be ported between operating systems, *e.g.* Mac OS, Linux and Windows, or even to network simulators (OPNET and NS2). Protolib in the simulation environments models a more realistic application-facing view of the underlying network protocols: *e.g.*, using Protolib, UDP, Multicast and TCP messages contain payload and therefore application data can be passed between NS2 routers, which is something not possible using unmodified NS2 implementations. However, to use Protolib, a routing protocol needs to be reimplemented to use the Protolib API and cannot use standard C calls for sockets and timers etc. Also, although Protolib exposes its API in Java, it does not support unmodified Java code, and any Java application would need to be rewritten to use this API instead.

In the Java world, there are three popular Java-based simulators. The first is the Java Network Simulator (JNS) [71], a Java implementation of the NS2 simulator. Just like NS2, it provides developers access to the simulation of networking protocols and then produces a trace file (same format as NAM trace files) which can be viewed in a network animator such as Javis [72]. As of 2010, it supports multicasting, multithreading and dynamic scheduling. J-Sim [73] is a real-time process-based simulation environment written entirely in Java, which provides a script interface in Perl, Tcl or Python.

Lastly, JiST [74] is a high-performance discrete event simulation engine that runs over a standard Java virtual machine. It is a prototype of a virtual machine-based simulation approach that attempts to unify traditional systems and language-based simulator designs.

In many ways, JNS, J-Sim, and other simulators provide similar environments to AgentJ. However, they suffer for two main reasons. First, they only support a fraction of the transport protocols and environments that the original NS2 framework implements, whereas AgentJ leverages the full stack of NS2 protocols. Second, these are Java-only systems and only support protocols that are written in Java and cannot be extended to support protocols in other languages, such as C++. In AgentJ, a protocol can be implemented in Java or embedded into NS2 in C++ and accessed from within AgentJ. Lastly, although JiST allows unmodified Java code, its current set of supported protocols and environments are limited, and neither J-Sim nor JNS allow unmodified distributed Java code to run within their simulator. A Java application has to be taken apart and structured into a discrete set of events that model the algorithm's behavior.

3.3 AgentJ Overview without Routing Functionalities

AgentJ [68] is an environment that facilitates the running of Java applications as *application agents* within the NS2 [59] discrete event simulator. By default, NS2 only supports simulations running either C++ or Tcl or a combination of the two. AgentJ builds on this environment to provide a linkage between the NS2 Tcl simulation orchestration commands and Java, which then in turn marshals the Java network, timing and System classes into Protolib C++. The various levels are shown in figure 3.1.

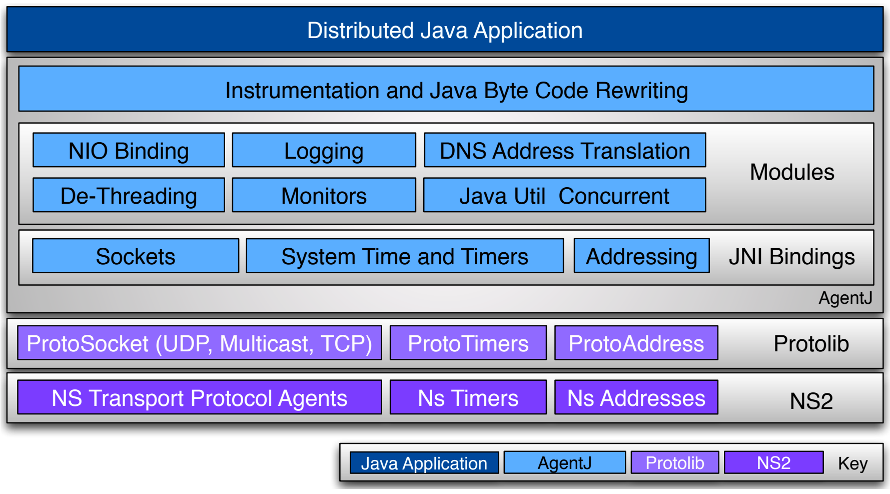


Figure 3.1: AgentJ architecture outlining its various components

The bytecode rewriting is contained within Java, *e.g.* Thread handling, NIO, Monitors, concurrency support, etc. However, network sockets, system time, timers and addresses are accessed by low-level calls using to the underlying C++ code as illustrated in figure 3.1. The implementation facilitates two-way transfers from Java to C++ and C++ to Java, allowing the creation of a Java Virtual Machine from the C++ side and for Java to talk to the underlying C++

objects. A callback system between the C++ (and Tcl) side of NS2 and Java is set up by AgentJ for interaction between NS2 and the Java implementation. To accomplish these interactions, AgentJ uses the Java Native Interface (JNI) [75] in order to connect the Java code to the C++ code of Protolib, which in turn maps to NS2, which allows NS2 to access of all Java classes, objects and methods. Thread handling in AgentJ is managed within the rewritten marshaled classes. Thread-synchronization is performed on the multiple Java threads so that NS2 waits either for all threads to finish or to block (waiting for data at a socket or a timer etc). This allows an unmodified Java multi-threaded application to be converted into a single-threaded Java application for simulation. For a detailed description of the rewriting mechanism of AgentJ, refer to [76].

3.4 Architectural Modifications of AgentJ for Support of Routing Protocols

This section details what architectural changes have been made to AgentJ in order to support running a Java routing protocol within NS2.

3.4.1 Overview

Figure 3.2 depicts the basic routing architecture of a Java application running on a single NS2 node over AgentJ.

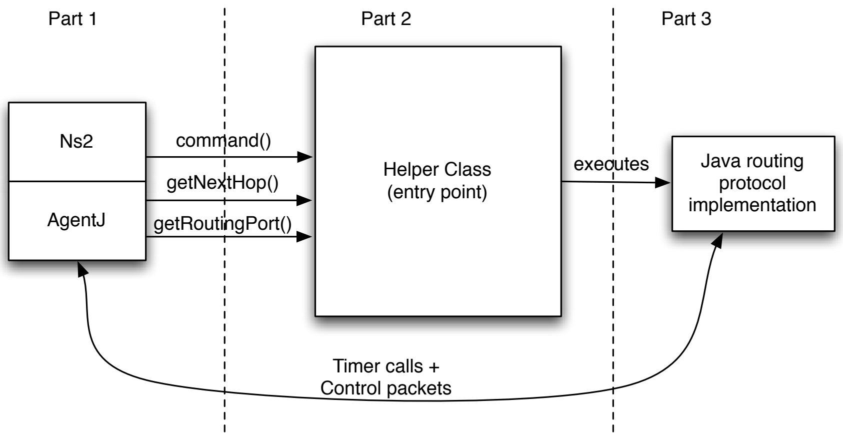


Figure 3.2: Architecture of an AgentJ agent attached to an NS2 node

Part 1 in the figure shows the NS2 node and the AgentJ agent that is attached to that node. This is already provided after installation of AgentJ and does not need any additional action by the user. Part 3, on the right side of the picture, depicts the Java routing protocol that the user wants to test in NS2. Since AgentJ automatically rewrites the Java bytecode, the Java routing protocol implementation does not need to be changed either. The *only* additional component, that the user needs to implement to "glue" the Java implementation to NS2, is depicted in part 2 in the middle: for launching the Java application and sending commands to it from NS2, it needs a single entry point.

3.4.2 Detailed Description

This section details the briefly presented architecture from section 3.4.1. Figure 3.3 depicts the detailed architecture of the modified version of AgentJ. On the left-hand side, the NS2 and AgentJ internal objects (`Agentj` and `AgentJRouter`) are depicted. These are implemented in C++ and use some Tcl code.

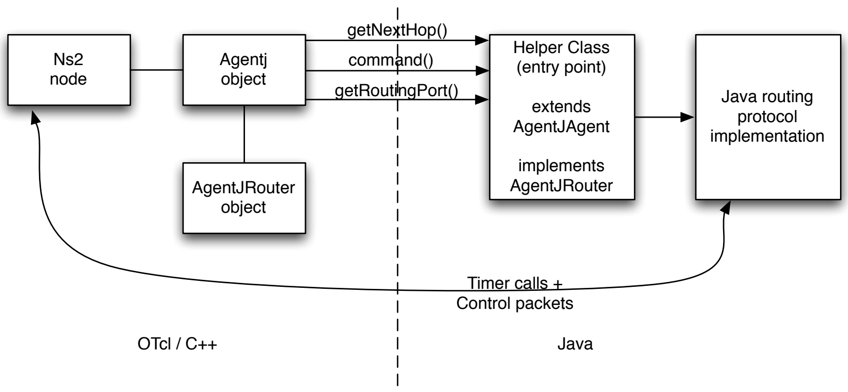


Figure 3.3: Modified architecture with an AgentJRouter interface

On the right side, the Java helper class that connects the Java routing protocol with AgentJ is displayed. This helper class is a subclass of `AgentJAgent` and implements the Java interface `AgentJRouter`.

In the original AgentJ code, the `Agentj` C++ agent was attached as an application agent to the NS2 node. In contrast, in the modified version of AgentJ with routing functionalities, the `Agentj` C++ agent can now also be attached to an NS2 node as a routing agent (*i.e.* it represents the RTR layer). The `AgentJRouter` C++ class treats packets that arrive at the RTR layer.

Incoming packets are received by the `Agentj` object, and then handed off to the `AgentJRouter` object (as illustrated in figure 3.4). If the packet is a control packet, it will be handed to a socket on the Java routing protocol. Otherwise, the packet will be treated by the demultiplexer as usual.

Whenever an outgoing data or control packet arrives at the RTR layer (*i.e.* the `Agentj` object), it is forwarded to the `AgentJRouter` object (as illustrated in figure 3.5). Then it has to be determined whether the packet is unicast or broadcast. If the packet is a broadcast packet, the destination is set to the `IP_BROADCAST` address and handed to the LL layer. If the packet is a unicast packet, the `getNextHop()` method on the Java routing protocol is invoked to determine the next hop address for the destination of the packet. If no next hop if found, the packet is

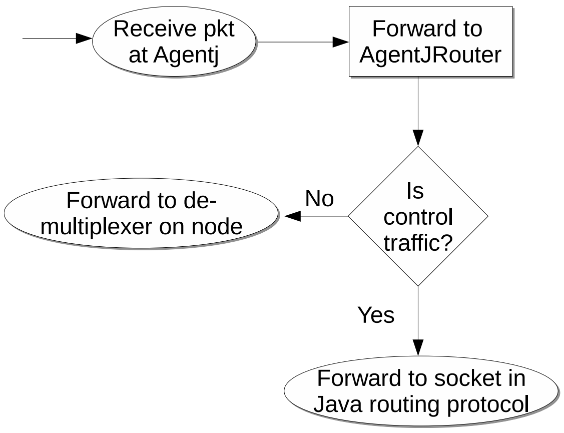

Receive pkt at Agentj
Forward to AgentJRouter
Is control traffic?
No
Forward to de-multiplexer on node
Yes
Forward to socket in Java routing protocol

Figure 3.4: Incoming packet on the RTR layer

dropped by NS2. This is reflected in the NS2 trace file as usual (DROP). Otherwise, the next hop is set to the returned next hop, and the packet is handed to the LL layer.

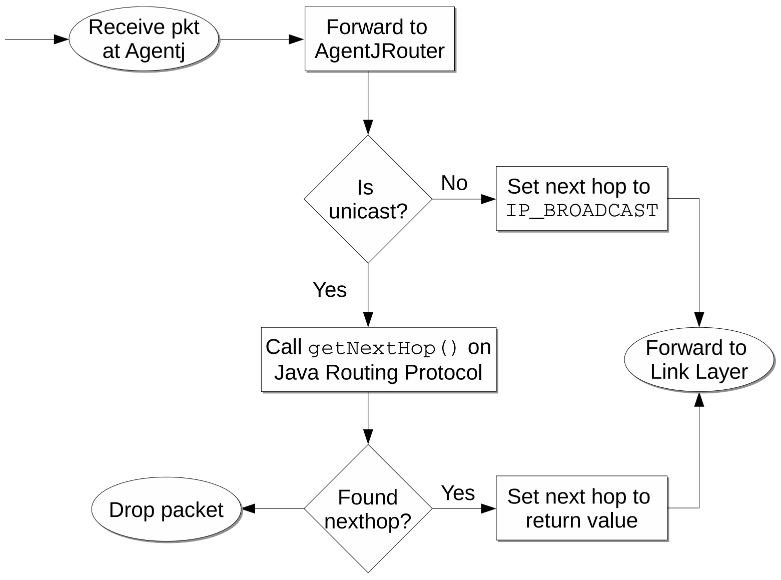


Figure 3.5: Outgoing packet on the RTR layer

Note that this represents the default behavior of `AgentJRouter` for incoming and outgoing packets, and may be overridden by the user, as described in appendix B.

3.5 Performance Comparison

In this section, a performance comparison between a Java protocol implementation using AgentJ and a C++ agent, is presented. The comparison investigates the time duration and memory consumption of running an NS2 simulation with the respective C++ and Java protocols. The simulation has been performed on a PC with an Intel Core2 CPU 2.13 GHz and 4 GB of RAM, with no other time- or memory-consuming processes running.

For this comparison, the same basic link state routing protocol has been implemented in both Java and C++. The C++ implementation is mainly based on Protolib [70]. In order to provide a fair comparison, both versions of the protocol have been implemented as similarly as possible, in terms of structure of the code as well as data structures. No complex data structures (such as trees, hash tables or priority queues) have been used in either version, only basic data types and lists. The algorithms used are exactly identical (*e.g.* identical complexity of Dijkstra). It can thus be assumed that both version of the protocol have very similar properties and are well suited for the comparison.

Figure 3.6 depicts the average time consumption for a single simulation run of the link state routing protocol from 20 to 80 routers, using the C++ implementation and the Java implementation respectively. Figure 3.7 shows the memory consumption of both versions. The simulations are averaged over 10 runs.

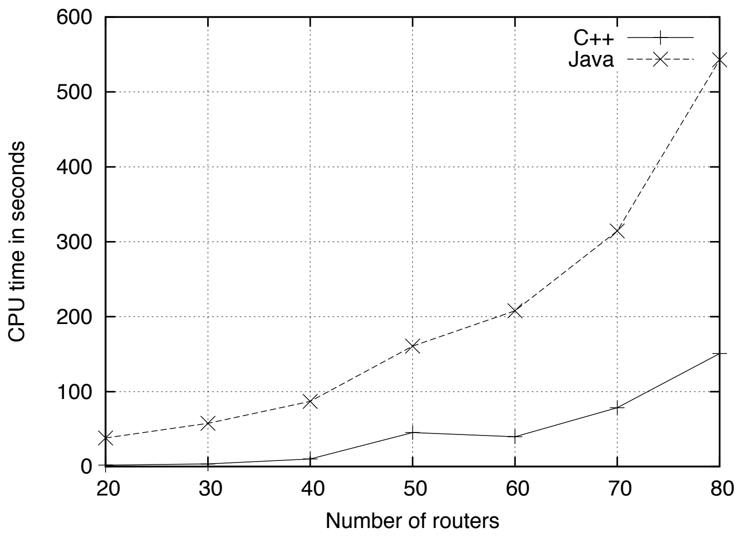


Figure 3.6: Average CPU time in NS2 with both a C++ and a Java implementation of the same link state routing protocol

As can be seen, the Java simulation takes more time and memory than the C++ implementation. This is due to several factors: First, it is commonly observed that Java is slower than C++ for identical tasks and consumes more memory, so the link state routing protocol itself will be slower. In addition, the thread linearization and bytecode translation for hooking Java commands into NS2 is costly.

It can be observed that the memory consumption difference between the Java and C++ run is almost constant, and the CPU time difference is proportional to the number of routers. Since commonly simulations are executed by a batch file and can run unattended, *e.g.* over night, the additional time consumption of Java protocols in AgentJ is in many cases acceptable.

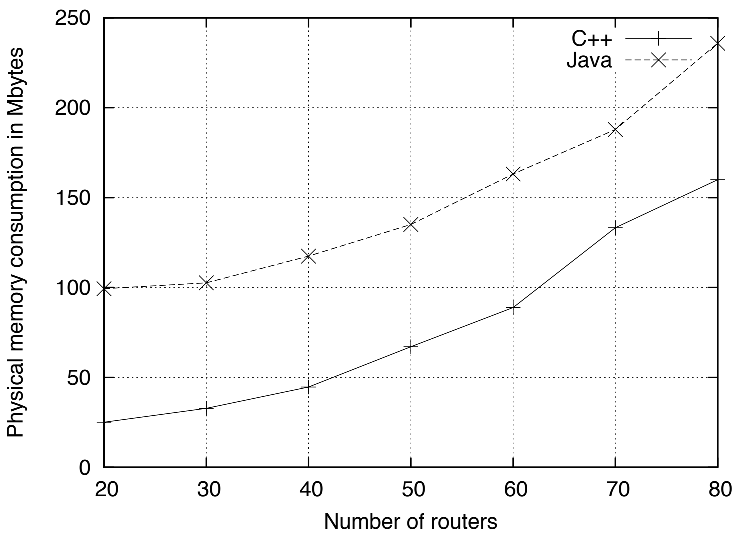


Figure 3.7: Average physical memory (non-swapped) consumption in NS2 with both a C++ and a Java implementation of the same link state routing protocol

3.6 Advantages of AgentJ Routing Framework

The basic version of AgentJ, as described in section 3.3 is limited to application-level agents in NS2 and does not support the execution of Java routing protocols. The modifications presented in section 3.4 extend AgentJ's applicability by allowing Java routing protocol implementations to be executed within the same environment in NS2. The main features of this routing framework are listed in the following:

- **Write once, run everywhere:** One of the major advantages of Java over other programming languages is that Java bytecode runs on all operating systems offering a Java Virtual Machine without change of code or recompilation. It can be assumed that the behavior is always the same, which is a useful property for protocol deployment. AgentJ with the routing extension allows to run a routing protocol implementation intended for a real network additionally in the network simulator NS2. It is thus possible to verify the correctness, as well as specific properties of the routing protocol implementation in the network simulator without rewriting part of the code.

- **Java offers an extensive API and is easy to write:** Java has many predefined classes that can be useful in the protocol development. In particular, existing data structures (such as trees, hash tables, priority queues, multicast sockets) allow a quick and easy development of routing protocols. In addition, no pointer arithmetics can lead to segmentation faults, and memory leaks occur much more rarely.

- **Add routing protocols without modification and recompilation of NS2:** For every C++ routing protocol implementation that is added to NS2, parts of the NS2 source code have to be modified, and NS2 has to be recompiled afterward. With AgentJ and the routing protocol extension, no modification of NS2 and subsequent recompilation is necessary. Multiple Java routing protocol implementations can thus be tested in parallel without modification of NS2, once AgentJ is installed.

- **Custom trace format for control messages:** Control traffic, that is exchanged between routers using a Java routing protocol implementation, is traced in the usual NS2 trace files. As such, all evaluation scripts that operate on these trace files, can be used without modifications. If additional tracing of the payload of control traffic packets is desired, this can be accomplished as well with a small modification of the CMU trace source code in NS2.
- **Custom behavior for treating messages on the routing layer:** For C++ routing protocol implementations, code has to be provided that treats incoming packets on the routing layer. For every such incoming packet, it must be decided whether this packet is a control packet or data packet, whether it needs to be forwarded or sent up to the application layer etc. In AgentJ with the routing extension, a default behavior is already provided, and thus needs not to be supplied by the routing protocol implementation. If a specific packet treatment is desired, however, the packet treating code can be easily extended.
- **Unmodified protocol execution:** Using other discrete time simulators does not guarantee that the actual algorithm is operating correctly because it does not run an exact copy of the actual deployed code. Thus, although a simulation may exhibit desirable properties, the implementation of it may not and may contain bugs or suffer from performance inefficiencies. By running the same actual Java code as in a real deployment with AgentJ, this not only allows accurate simulations to be performed, but it also provides a network debugging environment to analyze any overhead or implementation-specific content that might lead to inefficiencies or errors in the code.

3.7 Conclusion

This chapter has described how to run Java routing protocols within the network simulator NS2. The preexisting tool AgentJ allows for using Java agents on an NS2 node, but was not capable of instantiating routing agents. A modification of AgentJ has been presented in this chapter, which enables the use of Java routing protocols in NS2 without modification of the implemented Java routing protocol, adhering to the Java slogan “write once, run everywhere”. In addition, once AgentJ has been installed, NS2 does not need to be recompiled when a new Java routing protocol implementation is added. All parameters that need to be changed can be set in configuration files or in environmental variables, read at execution time. AgentJ, with the proposed routing extension, keeps full compatibility with NS2, meaning that it allows for output of all events in the usual NS2 trace files. Consequently, all evaluation tools used for parsing NS2 trace files can be used without modification. AgentJ simulations take more time and memory than C++. However, since commonly simulations are executed by a batch file and can run unattended, the additional time consumption of Java protocols in AgentJ is acceptable in many cases.

The routing functionalities have been included in the current distribution of AgentJ that is available for download at:

`http://downloads.pf.itd.nrl.navy.mil/agentj/nightly_snapshots`

Moreover, the contributions of this chapter have been published in [44].

Part II

General Architectural Considerations of Ad Hoc Networks

Chapter 4

Architectural Considerations of Ad Hoc Networks

MANET routing protocols operate with the assumption that MANET routers already have IP addresses and, if applicable, prefixes assigned – however no dedicated mechanisms for assigning IP addresses or distributing prefixes to MANET routers (called "autoconfiguration") have been standardized by the IETF. Academic literature has presented numerous such mechanisms, however that these may not necessarily have been developed with an explicit requirement of seamless integration with the Internet architecture and other IETF-related protocols.

It is worth analyzing the architecture of the Internet, a large and scalable computer network, as described in chapter 1, and to compare it with current architectural views in MANETs and autoconfiguration thereof. This chapter analyzes why existing proposals for autoconfiguration of MANETs are not compliant with the architecture of the Internet, and proposes an architecture which allows an autoconfiguration protocol to integrate with the Internet, and thus lays the foundation for MANETs to scale. Such a compliant autoconfiguration mechanism is then presented in chapter 5.

4.1 Chapter Outline

Section 4.2 outlines the notion of IP addresses and prefixes, as currently employed in the Internet, with the purpose of retaining the basic notions and properties of addresses and prefixes. Section 4.3 briefly recapitulates the architecture of MANETs as seen by the IETF. Section 4.4 discusses a commonly used – but misleading – terminology, and proposes an alternative terminology. Section 4.5 describes the goal of autoconfiguration mechanisms for MANETs. Section 4.6 describes which assumptions current autoconfiguration mechanisms in the Internet have. Section 4.7 argues why these mechanisms do not apply for the specific assumptions of MANETs. Section 4.8 concludes this chapter.

4.2 What is an IP Address?

Before elaborating on the specific issues regarding IP autoconfiguration of MANETs, as well as on existing IP autoconfiguration mechanisms, it is important to retain the basic notions of addresses – and their complements, prefixes. This section will therefore briefly describe the nature of addresses – which includes introducing the terms “address assignment” and “prefix delegation”. The discussions in this section are intended to provide an overview and motivation for the IP addressing architecture in the Internet, and not to be an exhaustive architectural discussion of the IP addressing architecture.

4.2.1 The Basics

In the most basic form, an IP address can be considered simply as follows:

- *an IP address is a unique identifier of a network interface.*

I.e., an IP address allows, uniquely, to address IP datagrams to one network interface among the many network interfaces present in the Internet.

According to this, the important property of an IP address is its network-wide uniqueness – and absent any other constraints, in order to uniquely identify any network interface on the Internet, this would be a sufficient property to ensure.

4.2.2 Networks

As described in chapter 1, the Internet is a big network, so a large number of IP addresses are required in order to ensure that each network interface can have a unique IP address. IPv4 offers 2^{32} unique addresses, and IPv6 offers 2^{128} unique addresses. If, as indicated in section 4.2.1, no other constraints than uniqueness were imposed, this would imply that each router in the Internet would have to maintain a routing table explicitly containing 2^{32} or 2^{128} entries – and, in order to construct its routing table, would have to exchange this much information with other routers. In order to reduce the amount of state information that each router would have to maintain (the size of the routing table) and exchange, constraints other than uniqueness are imposed on IP addresses as assigned throughout the Internet.

The basic idea in reducing the state that routers are to maintain and exchange is simple: assign IP addresses in an organized fashion, as indicated in the example in figure 4.1. Thus, all network interfaces with IP addresses from within a given interval would be attached to the same router – and the routers would each have to maintain and exchange state for each interval, rather than for each explicit IP address.

With 2^{32} or 2^{128} IP addresses, if the Internet was constructed uniquely as in the simple example in figure 4.1 illustrates, it would imply either a large number of routers, each with a still large number of network interfaces attached – or a small number of routers, each with even more network interfaces attached. A large number of routers still entails a large amount of state for each router to maintain and exchange – and a small number of routers each with a very

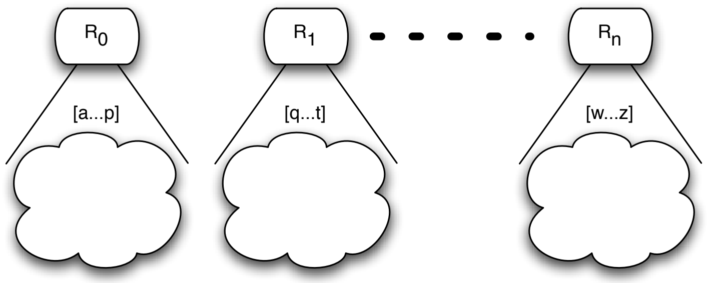


Figure 4.1: Flat topology of routers: each router having an address interval, with all network interfaces configured with addresses from within that interval being directly connected to that router.

large amount of network interfaces attached would concentrate traffic with ensuring capacity and reliability issues. Thus, and with respect to IP addresses, each of the clouds from figure 4.1 may itself contain routers, as illustrated in figure 4.2.

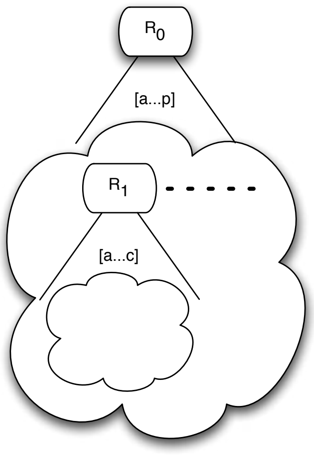


Figure 4.2: Hierarchical topology of routers: router R_0 delegates a subinterval [a...c] from its interval [a...p] to the subordinate router R_1

IP addresses within a cloud in figure 4.2 are assigned in the same organized fashion as in the example in figure 4.1, although now hierarchical: all addresses from within the interval [a-p] are in the cloud “under” router R_0; all the addresses from within the interval [a-c] are also “under” router R_1 etc.

The term used for giving a router an address interval and allowing that IP addresses from within that interval are attached to only *that* router is called *delegation.* Thus, an IP address assigned to a network interface indicates *where* in the hierarchy this network interface is located.

This implies two things:

- a router is *delegated* an IP address interval from someone, who has authority to do so – *i.e.*

from a router having (at least) that IP address interval delegated itself;

- two properties apply to IP addresses:
  - *an IP address is a unique identifier of a network interface* (as in section 4.2.1);
  - *an IP address implies a precise topological location in the addressing hierarchy.*

The last item is of special importance, since it implies that an address is not *just* an identifier, but that there is a semantics associated to an address – and that this semantics is important for the organization of the Internet.

4.2.3 Prefixes – the Internet Term for Intervals

Syntactically, an IP address is a simple sequence of bits – 32 bits for IPv4 and 128 bits for IPv6. In order to achieve an easy way of designating "IP address intervals" and "sub intervals" of such an IP address interval, prefixes are used. Structurally, a prefix is the n leftmost bits of an IP address, which are common to all IP addresses from within that interval, and n is said to be the *prefix length.* A classic example of a prefix and a prefix length from IPv4 would be `130.225.194/24`, where `130.225.194` is the prefix and `24` subsequently is the prefix length – indicating an IP address interval of `[130.225.194.0-130.225.194.255]`.

The general notation for a prefix in IPv6 is `p::/n`, implying that `p` is the prefix, the length of this prefix is `n`, and the `::` indicating that everything following the prefix (*i.e.* the bits other than the n leftmost bits of the address) is a sequence of zeros[1].

Thus, and with reference to figure 4.3, if the router R has the prefix `p::/32` assigned, subordinate routers R_1, R_2 and R_3 can be delegated sub-intervals by simply delegating these the prefixes `p:1::/64`, `p:2::/64` and `p:3::/64` respectively.

It shall be noted that:

- the prefixes thus delegated to these three routers are non-overlapping – *i.e.* the address ranges from which the routers R_1, R_2 and R_3 may respectively assign addresses are also non-overlapping;
- prefixes delegated to the routers R_1, R_2 and R_3 overlap with the prefix delegated to R_0, and thus the precise topological location of the routers R_1, R_2 and R_3 is subordinate to R_0.

Commonly, and for the remainder of this chapter, prefixes will simply be written as `p::` or `p:1::`, *i.e.* the prefix length will be omitted when written. The prefix length is in this case simply assumed equivalent to the length of the leftmost and explicitly written bits, *i.e.* that which appears to the left of the `::`.

[1]Although not relevant for the present discussion in this chapter, note that `::` in general in IPv6 indicates any number of consecutive four-digit groups (in hexadecimal) whose value is zero. Thus, `p::1` would be a valid way of representing an IPv6 address where the first bits are `p`, followed by one or more four-digit groups of zeros, followed by `1` ([77]).

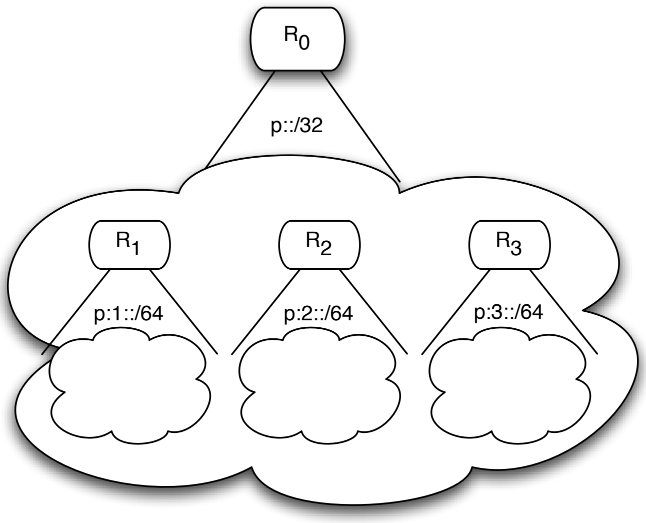


Figure 4.3: Prefix delegation: router R_0 has delegated non-overlapping subprefixes `p:1::/64`, `p:2::/64` and `p:3:/64` of its prefix `p::/32` to its subordinate routers.

4.2.4 Delegation and Assignment

Prefix delegation is, thus, the way in which a router acquires an IP address interval, which is topologically correct with respect to its location in the addressing hierarchy. From within this IP address interval, the router may – as detailed in section 4.2.3 – either delegate parts to subordinate routers, or assign addresses to network interfaces attached to it, as illustrated in figure 4.4. The important distinction in this is that a delegation of a prefix does not provide any network interface with a unique identity but rather allocates part of the IP address interval for exclusive use by a subordinate router – whereas address assignment implies that a network interface is configured with a specific unique and topologically correct IP address.

4.3 The MANET Architecture

In this section, the basic architecture relevant to a discussion on IP autoconfiguration for MANETs will be overviewed. In particular, the notion of a "link" as well as of a MANET router (MR) is important for the discussion in this chapter and are, therefore, presented in section 4.3.2 and section 4.3.4, respectively. In these sections, reference will be made to what can be called the *Classic IP Link Model*, which is therefore introduced in section 4.3.1.

4.3.1 The Classic IP Link Model

A core notion in the Internet is that of a *link*. A link, illustrated in figure 4.5, is assumed to have the following properties:

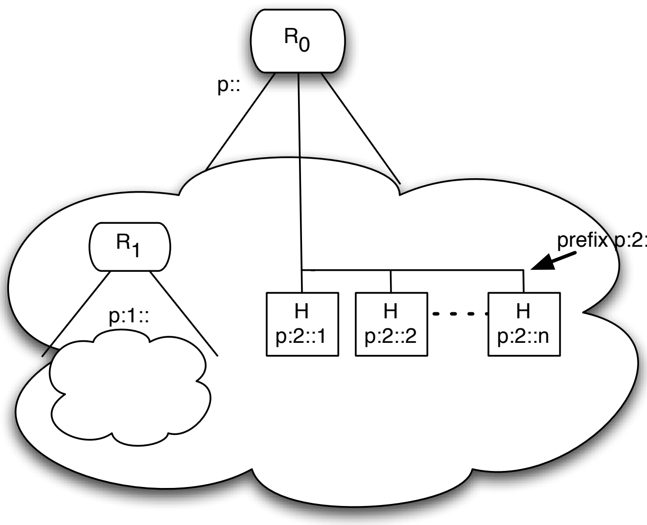


Figure 4.4: Prefix Delegation and Address Assignment: a router assigning IP addresses from part of its prefix to network interfaces directly attached, and delegating another part of its prefix to a subordinate router.

- IP datagrams are not forwarded at the network layer when communicating between network interfaces which are on the same link; hence
- TTL/hop-limit in IP datagrams are not decremented when communicating between network interfaces on the same link;
- IP datagrams with a TTL/hop-limit of 1 are (modulo data loss) delivered to all network interfaces on the same link and;
- link-local multicasts and broadcasts are received by all network interfaces on the same link – without forwarding.

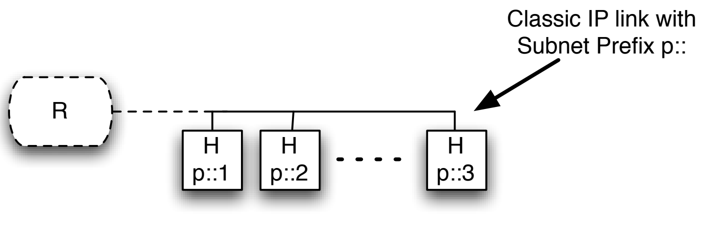


Figure 4.5: Classic IP Link Model: hosts (H) connected to the same link have assigned IP addresses from a common prefix, possibly assigned by a router (R).

When assigning an IP address to a network interface, this network interface is also configured with a prefix, such that the following constraint is respected:

- all network interfaces configured with addresses from within the same prefix `p::`, and with the same prefix `p::` assigned to the interfaces, can communicate directly with one another.

It follows from the above that the notion of "IP link" is tied with the notion of an "IP Subnet" (IPv4) or a prefix (IPv6), in that all network interfaces which are configured with the same subnet address or prefix are considered to be on the same IP link and thus that for communication between nodes within the same subnet, no forwarding is required and no decrement of TTL/hop-limit is performed.

Network interfaces within the same prefix or, for IPv4, within the same subnet, are within the classic IP link model assumed to also be attached to the same classic IP link as described above. For completeness, it should be mentioned that the inverse is not necessarily true: in some network configurations, interfaces connected to the same classic IP link may be configured within different prefixes or subnets.

4.3.2 The Missing Link

MANET interfaces are a specific class of network interfaces in that they exhibit what is commonly denoted as *semi-broadcast characteristics.* This implies that otherwise neighboring nodes may experience distinctly different local connectivity. As indicated in figure 4.6, the MANET interface of N2 is able to directly communicate with the MANET interface of N1 and N3 (*i.e.*, no forwarding, TTL/hop-limit is not decremented, a transmission from the MANET interface N2 is received by the MANET interfaces of both N1 and N3, including link-local multicasts and broadcasts).

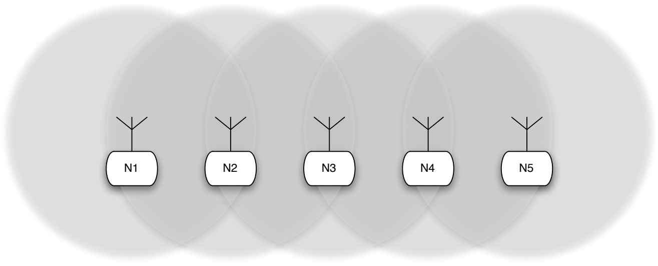


Figure 4.6: MANET: nodes (N) with MANET interfaces. The light gray area indicates the coverage area of each MANET interface.

Considering the properties listed for an IP link in section 4.3.1, this might imply that the MANET interfaces of N1, N2 and N3 would be connected to the same link. However the semi-broadcast nature of MANET interfaces implies that this is not so: transmissions from N1 will not reach N3 without forwarding – which entails that TTL/hop-limit is decremented, and that link-local multicast and broadcast from the MANET interface of N1 will not reach the MANET interface of N3.

The semi-broadcast nature of MANET interfaces implies that:

- any two MANET interfaces can not be assumed to be connected to the same link; thus

- the IP address / prefix configuration of MANET interfaces must be such that their configuration (as per the constraint in section 4.3.1) does not indicate that they can be assumed to be connected to the same link.

A rationale for the latter of these two implications is given in the following section.

4.3.3 /32 and /128 Prefix Lengths

In early MANET deployments, a common occurrence was to configure a MANET as "a subnet" and configure it as indicated in figure 4.7: the MANET would have a subnet prefix, say 192.168.1.0/24 and MANET interfaces would be configured with this subnet prefix, *e.g.* 192.168.1.3/24.

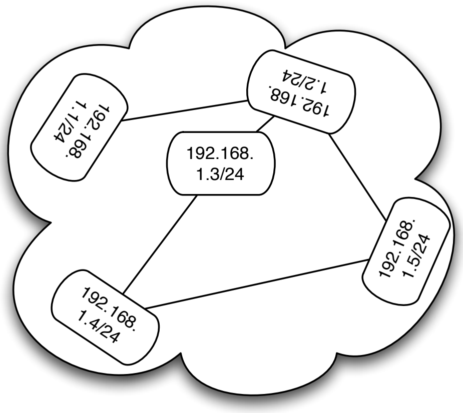


Figure 4.7: Misconfigured MANET: MANET interfaces configured with a shared /24 prefix, causing router 192.126.1.2 to produce an ICMP redirect when forwarding IP datagrams from 192.126.1.1 to 192.168.1.3.

If, for example, a data packet was transmitted by the MANET router 192.168.1.1 to be received by 192.168.1.3, then this would – with reference to figure 4.7 – have to be forwarded by the MANET router 192.168.1.2. With the MANET interfaces in this MANET being configured with the subnet prefix 192.168.1 and the prefix length /24, it was observed that this produced ICMP redirects by 192.168.1.2.

An early, and often suggested, solution was to "treat the symptoms rather than cure the disease" by disabling ICMP redirects on MANET routers[2] – *i.e.* to require modifications to the IP stack operation in order that it can be supporting MANETs.

The ICMP redirect is intended to be used to inform a source to send packets using an alternative, more direct, route – *e.g.* if a source, s wishes to send a data packet to a destination node d via the path *s-R1-R2-d*, and if *R1* knows that a direct path *s-R2-d* is available, then the ICMP redirect from R1 will inform s of this route.

[2]For example, the olsr.org MANET routing protocol implementation [78] disables ICMP redirects

Two interfaces, configured with addresses from within the same subnet prefix, and with the same prefix length are defined to be reachable from each other within a single IP hop. More precisely, it is assumed that all interfaces with IP addresses within that subnet prefix and configured with the same prefix length are directly reachable from each other without forwarding by an intermediate router. Hence, a way for *R1* to know that a direct path may exist between *h* and *R2* is if *h*, *R1* and *R2* are configured with IP addresses from within the same subnet prefix and within the same subnet prefix.

Returning to the MANET scenario in figure 4.7, with all MANET interfaces being configured with the same subnet prefix and the same prefix length, it follows from the discussion above that all these MANET interfaces are expected to be directly reachable from each other within a single IP hop. When, in this configuration, the router 192.168.1.2 is requested to forward a data packet from 192.168.1.1 to 192.168.1.3, it is expected that it generates an ICMP redirect to 192.168.1.1 suggesting that a direct path exists from 192.168.1.1 to 192.168.1.3 – as this is what the configuration suggests.

Rather than "treating the symptoms" and disabling ICMP redirects, requiring /32 and /128 prefix lengths on MANET interfaces "cures the disease". An interface so configured will not make any assumptions about other interfaces being within a single IP hop, and so will not generate ICMP redirects when forwarding traffic.

4.3.4 Definition of a MANET Router

A MANET router is a router having, at least, one MANET interface – *i.e.* an interface exhibiting semi-broadcast characteristics as indicated in section 4.3.2 – and otherwise retaining the usual characteristics of a router. In particular, a MANET router:

- may have a prefix delegated to it;
- may delegate (part of) that prefix to other subordinate routers;
- may have networks attached to it, in particular these networks may be of any type, (including, but not limited to, other MANETs), thus respecting usual routing and addressing hierarchies.

4.3.4.1 IP Hop Isolation

Upper-layer applications and protocols are designed with a set of (often not explicitly expressed) assumptions as to the nature and characteristics of the underlying network communications fabric. This includes, but is not limited to, assumptions on (i) the relationship between a "subnet" and the ability for interfaces configured within the same subnet to all be able to communicate with each other directly, *i.e.* in one IP hop (as evoked in section 4.3.1), (ii) that communication between interfaces on the same link is symmetric (if interface A can receive packets from interface B, then interface B can receive packets from interface A), (iii) that link-local multicast (*i.e.* with TTL or hop-limit equal to 1) is supported and is well defined (*e.g.* that the recipients of a link local multicast from interface A is identical to those of interface B, assuming that interfaces A and B are on the same link). Essentially, upper-layer applications and protocols assume an underlying link with characteristics similar to those of an Ethernet.

However, as described in section 4.3.1, the semi-broadcast nature of MANET links expose different characteristics than an Ethernet link. There are, essentially, four potential ways of addressing this problem: requiring all upper-layer applications and protocols to become “MANET aware”, inventing mechanisms for presenting a MANET as-if it was an Ethernet, pushing the problem to layer 2, or encapsulating any MANET specific behavior in a way such that it is only exposed to explicitly MANET aware applications and protocols.

Requiring all upper-layer applications and protocols to be reworked such that they be MANET aware is impossible, due to the sheer volume of such, is undesirable as a tenant of the Internet is to provide an abstraction for applications from the interconnect characteristics, and would likely, not be without technical problems of its own – for example, section 4.3.3 exposes the complexity with respect to the single problem of correctly ensuring the relationship between “subnet” and which MANET interfaces are able to directly communicate.

Pushing the problem to layer 2, and requiring a layer 2 to present an “Ethernet-like” medium would for a given layer 2 solve the problem – but do away with the ability to deploy heterogeneous MANETs, and would be akin to e.g. requiring OSPF to do away with all but one interface type and require all layer 2’s over which OSPF is expected to run to adhere to this single remaining interface type.

The last way of addressing this problem is to encapsulate MANET specific behaviors in a way such that it only is exposed to explicitly MANET aware applications and protocols. This approach has several merits, not the least of which is that this is exactly the way in which existing routing protocols are deployed: interfaces between routers, and their characteristics, are exposed only to applications and protocols which are explicitly aware of these interfaces and their characteristics, with all other applications and protocols being isolated from these by being connected via an “Ethernet-like” link to the router – *i.e.* by virtue of being isolated behind an IP hop (the router).

For example, applications on hosts are exposed to an “Ethernet-like” link with other hosts and potentially routers – and are blissfully unaware whether these routers are connected using point-to-point links, NBMA links, broadcast links etc.

For these reasons, configuring a MANET such that the MANET router is isolating the MANET interfaces and their characteristics from hosts and other (non-MANET) networks by an IP hop, and requiring that on MANET routers, only explicitly MANET aware applications are exposed to MANET interfaces, is a safe and simple approach that entails no additional complex mechanisms and signaling. This also requires no changes to neither the IP stack nor the upper-layer applications and protocols operating on these hosts and (non-MANET) networks.

4.3.5 ‘M’ for Mobility – ‘A’ for Ad hoc

As indicated in section 1.2.1, MANETs are characterized by a dynamic topology – *i.e.* that the network topology, in particular the set of participating MANET routers as well the ability for two MANET routers to communicate among themselves may change over time. In particular, this entails that a formerly single MANET may separate into two independent MANETs, and that two formerly independent MANETs may merge into one single larger MANET.

Additionally, the ‘A’, for ad hoc, implies that the network may emerge spontaneously and, potentially, without any pre-determined infrastructure elements (or without any such elements altogether) such as a router providing access to an outside network such as the Internet. This implies that a MANET may appear either in isolation, where the *uniqueness property* of IP addresses is the sole priority, configuration-wise – or in a situation where it may be integrated/attached to an outside network where, in addition to the *uniqueness property*, IP addresses / prefixes should correspond to the topological location of the MANET within the addressing hierarchy.

This latter point is somewhat analogous to other IP networks, in which a collection of hosts may connect with each other in a private network without connection to outside networks and wish to acquire IP addresses where the *uniqueness property* of IP addresses / prefixes is the sole priority (link-local addresses) – as well as be part of a network where in addition to the *uniqueness property*, IP addresses / prefixes should correspond to the topological location of the network interfaces within the addressing hierarchy (global addresses). As will be detailed in section 4.6, these two are both supported in the Internet via IPv6 Stateless Autoconfiguration. Moreover, an analogy to MANETs exist, as indicated in that section.

4.4 Connected vs. Disconnected MANETs

Occasionally, the terms *“connected MANET”* and *“disconnected MANET”* (or, for the latter also *“stand-alone MANET”*) are employed when describing different deployments scenarios (for example in [79, 80, 81, 82, 83]). These terms are misleading and unfortunate; this section will show why they are wrong in section 4.4.1, why they are unfortunate in section 4.4.2 – and propose an alternative and correct way of describing different MANET deployment scenarios in section 4.4.3.

4.4.1 On why these Terms are Misleading

In order to correctly analyze why the terms *“connected MANET”*, *“disconnected MANET”* and *“stand-alone MANET”* are misleading, consider first the scenario depicted in figure 4.8. This MANET contains four MANET routers (MRs), each of which having one MANET interface and one additional network interface towards a classic IP link. MR1 has, on this classic IP link, a (non-MANET) router, R, which has another network interface towards another classic IP link. There are no components (other than, perhaps, hosts) in this network, other than those depicted in figure 4.8.

The obvious question to ask, then, if this MANET in figure 4.8 is a *“connected” MANET* or if it is a *“disconnected MANET”*?

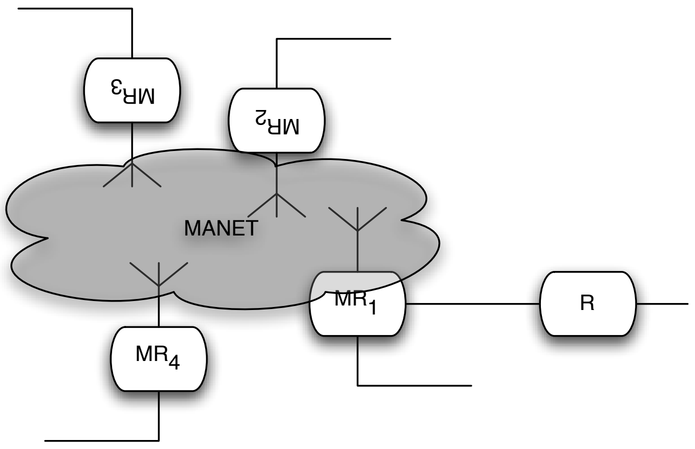


Figure 4.8: A MANET: four MANET routers (MRs), with MR_1 connected via a classical IP link to a non-MANET router R. Is this a *"connected MANET"*?

4.4.1.1 Graph Theory?

Being "connected" may refer to a property of a graph, recalling that a *connected graph is an undirected graph that has a path between every pair of vertices.*

A MANET is intended – even expected – to possess this property, which the MANET in figure 4.8 certainly does posses.

- Thus, if the intended semantics of a *"connected MANET"* is with reference to graph theory, then the MANET in figure 4.8 indisputably is a *"connected MANET"*.

4.4.1.2 Outside Networks?

The common use of the term *"connected MANET"*, however, seems intended to carry a semantics that go beyond that of graph theory. Assuming that MANETs are supposed to be composed of a connected graph of MANET routers, a MANET being *"connected"* may be in reference to the existence of links from MANET routers over non-MANET interfaces to other networks. Considering the scenario in figure 4.8, this MANET indisputably is connected to another network over a non-MANET interface – the easiest example hereof is the existence of a connection from the MANET via MR1 to R.

Considering an even simpler MANET, such as the one depicted in figure 4.9 where a single MANET router, MR_1, has a single MANET interface and a single network interface towards a classic IP link, the MANET can be said to be composed of the single MANET interface – which, indeed is connected to another network, indicated by the gray oval, via MR_1.

- Thus, if the intended semantics of a *"connected MANET"* is with reference to the existence links connecting the MANET to other networks, then the MANET in figure 4.8 indisputably is a *"connected MANET"*. Moreover, a single MANET router with a single

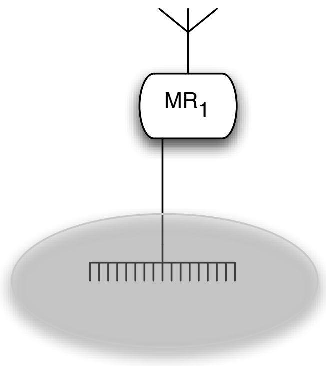


Figure 4.9: A Minimal *"connected"* MANET: the mobile router MR_1 has a single MANET interface and a single network interface towards a classic IP link.

MANET interface and a single interface connected to a classic IP link is indisputably a *connected MANET* – and, indeed, so is every possible MANET.

4.4.1.3 The Internet?

It may be that the intention of the term *"connected MANET"* is to indicate if a MANET is connected to "The Internet", with the term *"disconnected MANET"* being used otherwise. There are a number of problems inherent in this usage, since:

- "The Internet" as such does not exist as a unique and identifiable entity, but is rather a concept of "a network of networks" [84]. Thus, what does it mean to be connected to the Internet? That one can access a particularly common www search-engine? Are "Internets" (other than the one where this particularly common www search-engine exists) then not "Internets"? If not, then what, other than arbitrary existence of a given service, distinguishes the "real" and the "other" Internets.

- Even if a unique "Internet" could be identified and defined, this would not suffice, since:
  - a router which connects to another router which is part of this "Internet" does not become "connected to" the Internet – but becomes "part of" the Internet. A MANET router connecting via a network interface to a classic IP link and thereby to another router which is "part of" the "Internet" therefore also becomes "part of" this "Internet".
  - specifically, routers in the "Internet" are not segregated into classes of "Internet routers" and "Internet routers, connected to the Internet";
  - as such, routers do not signal if they are indeed "Internet routers" or not – implying that the scenario in figure 4.8 would be indistinguishable from the scenario where the outside network was this "Internet".

- Thus, if the intended semantics of a *"connected MANET"* is with reference to the existence of links connecting the MANET to the "Internet", then if a given MANET is a *"connected MANET"* or not is indeterminable.

4.4.2 On why these Terms are Unfortunate

The use of the terms *"connected MANET"* and *"disconnected MANET"* indicates a desire for classifying MANETs into two categories, with networks of either category having a clearly defined and easily identifiable set of properties, distinct from networks of the other category.

The fact that since a single MANET router with a single MANET interface and a single network interface connected to a classic IP link, as well as more complex MANETs can be said to be a *"connected MANET"*, as per both section 4.4.1.1 and section 4.4.1.2, indicates, that all MANETs are, indeed, in the *"connected MANET"* category. Or, in other words, the clearly defined and easily identifiable set of properties that would set a *"connected MANET"* and a *"disconnected MANET"* apart are not to be found in neither graph theory nor in the existence of links from the MANET to outside networks.

Since, as per section 4.4.1.3, if a MANET, or more generally "any router" is "connected to the Internet" or not, is indeterminable, using this as a classifying metric is unfortunate for the same reasons: it does not provide a well-defined set of criteria for determining if a MANET is a *"connected MANET"* or not.

All of the above illustrates that these proposed classifications and terms are unfortunate, or perhaps unnecessary, since the terms *"connected MANET"*, *"disconnected MANET"* and *"stand-alone MANET"* do not clearly specify scenarios in which distinct problems are present and therefore where distinct considerations are required.

4.4.3 A Better Taxonomy

The terms *"connected MANET"* and *"disconnected MANET"* have been proposed for a reason – a reason which is valid enough, but which needs to be clearly described and, consequently, a valid terminology needs be defined.

In order to understand the issue, it is important to recapitulate the properties of an IP address, as described in section 4.2 and repeated below:

- *an IP address is a unique identifier of a network interface* (as in section 4.2.1);
- *an IP address implies a precise topological location in the addressing hierarchy.*

Of particular importance for the discussion in the following is the second of these two properties. Considering the two scenarios in figure 4.10 and figure 4.11 for illustrating the importance this makes.

4.4.3.1 Autonomous MANETs

In figure 4.10, the MANET exists, with a number of connections to other networks. The particularity of this scenario is, that none of these networks imposes an addressing hierarchy on the

MANET. To make reference to the tree-like figures in section 4.2, the MANET is the "root" of the tree, with routers in the other networks being subordinate to the MANET. As such, these non-MANET routers should have prefixes delegated to them from the MANET, such that these prefixes reflect the non-MANET routers topological location below their upper MANET routers.

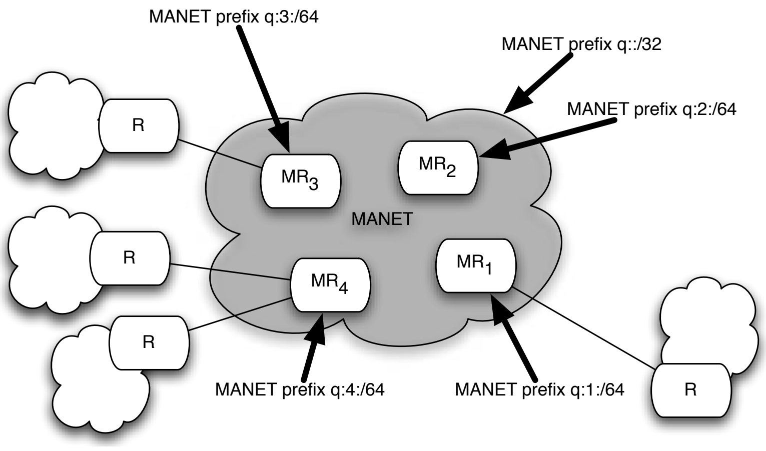


Figure 4.10: Autonomous MANET: The MANET is the "root" of the addressing hierarchy

4.4.3.2 Subordinate MANETs

In figure 4.11, the MANET exists, with a number of connections to other networks. The particularity of this scenario is, that one of these networks, the one denoted "N", imposes an addressing hierarchy on the MANET. To make reference to the tree-like figures in section 4.2, the MANET is an "internal node" of the tree, with the router R in the network N. Thus, to ensure that the second property of IP addresses (that they be correct with respect to a topological location in the addressing hierarchy imposed by the network A), IP addresses within the MANET must be assigned from within a prefix delegated to R – possibly by R delegating part of its prefix for use within the MANET.

As with the "Autonomous MANETS", described above, non-MANET routers in other networks, subordinate to the MANET, should have prefixes delegated to them from the MANET, such that these prefixes reflect the non-MANET routers topological location below their upper MANET routers.

On the topic of subordinate MANETs, such may be *multi-homed*, *i.e.* be subordinate to two or more routers, each of which impose their own topological location in an addressing hierarchy. This situation is no different than that of any other multi-homing situation, in that the MANET may acquire a prefix from either or both of these, and assign addresses from either or both of these – so long as it is ensured that traffic from the MANET which is generated with a given source IP address is transited through the appropriate router (*i.e.* the one from whose prefix that source IP address is acquired).

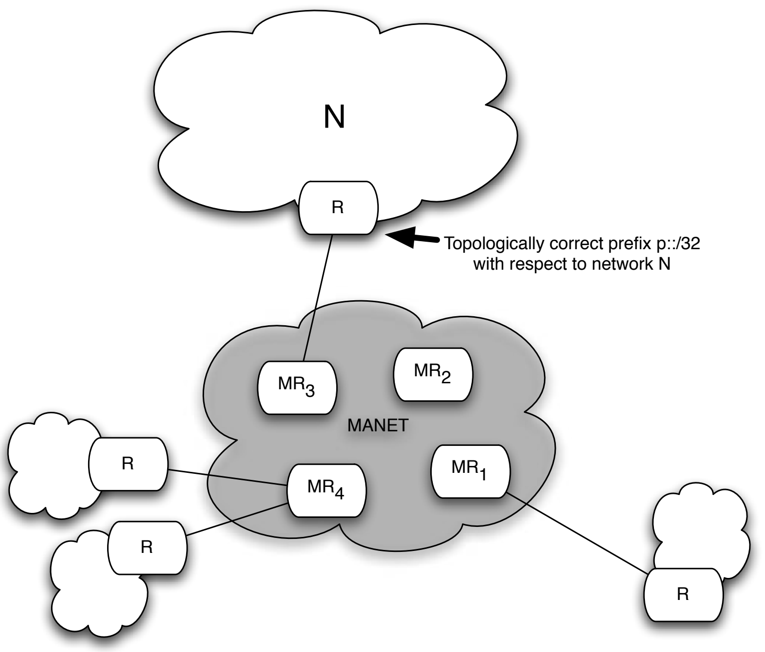

N
R
Topologically correct prefix p::/32
with respect to network N
MR$_3$
MR$_2$
MANET
R
MR$_4$
MR$_1$
R
R

Figure 4.11: Subordinate MANET: The MANET is imposed an addressing hierarchy by a superordinate router

4.5 AUTOCONF – The Goals

With the general considerations on IP addresses and prefixes in section 4.2, and the terminology in section 4.4.3 identifying two distinct MANET deployments, the goals that a MANET AUTOCONF solution should satisfy can be summarized into a simple set of goals:

- provide each MANET router with a unique prefix;
- respecting addressing hierarchies and the Internet addressing architecture;
- contain any MANET-specific behaviors to the MANET interfaces of MANET routers.

This section will detail these goals a bit further, starting with general architectural considerations in section 4.5.1, followed by specific considerations for Autonomous MANETs and Subordinate MANETs, in section 4.5.2 and section 4.5.3, respectively. Each of these sections will specify the precise goal for an AUTOCONF solution for each of these two MANET classes.

Section 4.5.4 will reflect on the architectural consequences of this approach, specifically with respect to that which applies to the MANET interfaces.

4.5.1 Architectural Considerations

The overall goal of AUTOCONF – AUTOmatic CONFiguration for MANETs – is to ensure that network interfaces can be uniquely identified, respecting the addressing architecture in section 4.2 and the constraint outlined in section 4.3.1. In other words, that network interfaces are automatically configured, respecting the Internet and the IP architecture.

In a MANET router, this entails that the MANET interface is to be configured with an IP address and prefix – and that network interfaces attached to a given MANET router, as well as routers subordinate to a given MANET router are to be, respectively, assigned IP addresses and delegated prefixes from within a unique prefix over which that MANET router is authoritative. This implies that:

- each MANET router must acquire a unique prefix, over which it is authoritative.

Under which constraints this prefix is to be acquired will be detailed in subsequent sections. With each MANET router having acquired such a unique prefix, the following considerations apply:

- a MANET router behaves, with respect to subordinate routers and networks, identical to any other router, in particular in respecting the addressing hierarchies;
- given that the MANET router has acquired a unique prefix, the MANET interface can be configured with an address from within this prefix and a prefix length of /128 (for IPv6) or /32 (for IPv4).

4.5.2 Acquiring a Prefix in an Autonomous MANET

The particularity of an Autonomous MANET is the absence of an externally imposed addressing hierarchy – and thereby the absence of an entity which can provide unique and topologically correct prefixes to MANET routers.

In order for each MANET router to acquire a unique prefix, an AUTOCONF solution must therefore be developed that:

- allows MANET routers to, from within a well-known prefix similar to the well-known Link Local Prefix, and in a distributed fashion, each acquire a prefix which is unique within this MANET;
- addresses and prefixes from within this well-known prefix are characterized by being unique only within one MANET.

4.5.3 Acquiring a Prefix in a Subordinate MANET

The particularity of a Subordinate MANET is the presence of one or more externally imposed addressing hierarchies – and thereby the presence of one or more entities which can provide unique and topologically correct prefixes to MANET routers.

In a Subordinate MANET, the solution detailed for Autonomous MANETs may be applied in order to acquire MANET-wide unique and local prefixes for each MANET router. In addition, an AUTOCONF solution must be developed that:

- allows MANET routers to request delegation of a prefix from one or more of the externally imposed addressing hierarchies – a prefix or prefixes which will be correct with respect to the MANETs topological location within these addressing hierarchies.

4.5.4 Consequences

Recalling the tree-like addressing hierarchies from section 4.2, the set of MANET routers in a MANET appear as a single "internal node" in the tree. This means that:

- "downwards" in the tree, *i.e.* to connected networks and subordinate routers, the addressing hierarchy is respected with respect to each MANET router;
- "upwards" in the tree, if the MANET is a Subordinate MANET, the MANET routers will respect the addressing hierarchy with respect to the external entity(ies) from which prefixes have been acquired;
- "horizontally" within the MANET, and due to the potentially dynamic topology of the MANET, no permanent relationship – and, in particular, no hierarchy – is ensured between MANET routers. Negative consequences hereof are prevented by configuring the MANET interfaces such that their configurations do not allow assumptions that they all be connected to the same "link", as detailed in section 4.3. This particularity is exposed *only* horizontally between MANET routers, and is neither propagated upwards or downwards in the tree.

4.6 Automatic Configuration in the Internet

Other than assigning IP addresses and prefixes to network interfaces in a static manner, there are several mechanisms of automatic configuration. Multiple solutions have been standardized by the IETF and are wide-spread in implementations. In this section, an overview of the most common autoconfiguration mechanisms is presented. Among them are DHCP (presented in section 4.6.1), and Stateless Autoconfiguration (refer to section 4.6.2).

These mechanisms rely on certain assumptions of the Internet, some of them being mentioned in section 4.6.3.

4.6.1 Dynamic Host Configuration Protocol (DHCP)

DHCP ([22] for IPv4, [23] for IPv6) enables automatic allocation of an IP address to a host by a DHCP server. A host requiring an IP address contacts a DHCP server and requests a new address. The DHCP server will dynamically assign an address from a certain interval of addresses, and allocate a so called "lease" of that address to the client. The client can then use the address for a certain time. If it wants to keep the address for a longer time, it has to prolong the lease. If the DHCP server is not in the same subnet as the DHCP client, it is possible to use a DHCP relay agent which can forward the messages to a different IP network.

For automatically delegating IPv6 prefixes to routers, there exists an option in a DHCP message called DHCP-PD (as specified in [85]). A router may demand a prefix from another router by sending a DHCP request including the DHCP-PD option. The DHCP server may then delegate a subordinate prefix of its own prefix to the client. The prefix delegation option may also be relayed through a DHCP relay agent.

4.6.2 Stateless Autoconfiguration / Neighbor Discovery Protocol

Stateless Autoconfiguration (SLAAC) (as specified in [21]) is another way of assigning IP addresses to hosts which has been introduced with IPv6 (and cannot be applied in IPv4). In contrast to DHCP, a host requiring an address does not need to contact any kind of server. Instead, the protocol is completely distributed.

A host first selects a tentative IPv6 address by attaching its host identifier (in most cases its MAC address) to the well-known link-local prefix. It then has to verify that no other host in the same subnet has the same address by broadcasting NDP messages [86] to the link. If the address is not unique, the autoconfiguration process will be aborted. Upon a successful address uniqueness test, a host may request a prefix from any router on the link by an exchange of NDP messages. It will again attach its host identifier to that router prefix and possibly repeat the address uniqueness test.

4.6.3 IP Architecture Assumptions

The two mechanisms presented in the sections 4.6.1 and 4.6.2 are based on certain assumptions which hold in the Internet. Most importantly, these assumptions are:

1. Messages (*e.g.* DHCP Request, NDP) are link-locally broadcast with a TTL of 1, and may thus never be forwarded by a router.

2. The local topology is relatively stable.

3. Routers acquiring a prefix from a delegating router are in a subordinate topology with respect to that router.

An example for assumption 1 are NDP messages. They have a TTL of 1, and whenever a node receives an NDP message with a decremented TTL, it will reject the message, and the uniqueness test will fail. Thus, NDP messages may never be forwarded by a router.

Assumption 2 means that in classical IP networks, links will not break down very frequently, and the topology of the LAN will not change in an ad-hoc manner. This assumption is for example important for DHCP where all clients are required to reach the DHCP server to acquire or prolong their lease. If the link is broken, the allocated address will be invalidated after the lease has expired.

Assumption 3 applies for instance for DHCP-PD. The requesting router is topologically subordinate to the delegating router, as depicted in figure 4.2.

4.7 Internet vs MANETs

As presented in section 4.6, there are several well-working standards for autoconfiguring IP addresses and prefixes in the Internet. These are all based on certain assumptions that are valid in the architecture of the Internet, some of them having been presented in section 4.6.3. However, as shown in section 4.3, MANETs have some different architectural properties in comparison to classical IP networks. These special properties avoid the verbatim usage of the presented autoconfiguration mechanisms, which will be demonstrated in this section.

4.7.1 Different Link-Model of MANETs

One reason why the autoconfiguration methods presented in section 4.6 cannot directly work on MANETs is the different link model of MANETs in comparison to the classic IP link model (refer to section 4.3.1 and 4.3.2 for a more detailed description of the link model in the MANET architectural model). In the classic IP link model, a link-locally broadcast packet will never be forwarded by a router. However, to assure that MANET router prefixes are non-overlapping within a certain scope, uniqueness of router prefixes must be assured over several hops. This invalidates the usage of NDP for the prefix uniqueness test, as the NS and NA messages would have to be forwarded by routers.

4.7.2 Routers vs. Hosts

Several proposals for MANET autoconfiguration in literature suggest to simply use NDP messages to configure MANET nodes. However, this contradicts the MANET architecture as proposed in section 4.3 where MANET nodes are described as routers and possibly attached nodes. [21] specifically states: "*The autoconfiguration process specified in this document applies only to hosts and not routers*". The reason for this is that one main goal of autoconfiguration is to allocate prefixes to MANET routers (refer to section 4.5) and not IP addresses to hosts. It has to be assured that these prefixes are non-overlapping so that hosts attached to that router can then be autoconfigured using classical autoconfiguration mechanisms as presented in section 4.6.

4.7.3 Mobility in MANETs

One main difference of MANETs in comparison to classical wired networks is the mobility of routers and thus a dynamic network topology, as described in chapter 1. Routes between MANET routers may change very frequently and may also break down. When using DHCP for prefix delegation, it is important that all clients can reach the DHCP server which is not guaranteed in a MANET: the router running a DHCP server may be cut off from the MANET. Thus, no new router would ever get a prefix, and as soon as the leases of prefixes expire, participating routers in the MANET will lose their prefixes. In addition, neither DHCP servers nor NDP can handle merging and partitioning of MANETs.

4.7.4 Relationship between MANET Routers

As seen in the example in section 4.6.3, in the Internet, routers acquiring a prefix from a delegating router are compulsorily in a subordinate topology with respect to that router. This is not always true in a MANET. While MANET routers may allocate a subprefix from a superordinate router, they are not forced to. This case is depicted in figure 4.12, where two routers in a MANET with a prefix `p:1::` and `p:2::`, respectively, are on the same hierarchical level rather than in a subordinated topology. This is the reason why delegating prefixes using DHCP-PD will not directly work because `p:2::` is not subordinate prefix of `p:1::` and can thus not be delegated by MR_1.

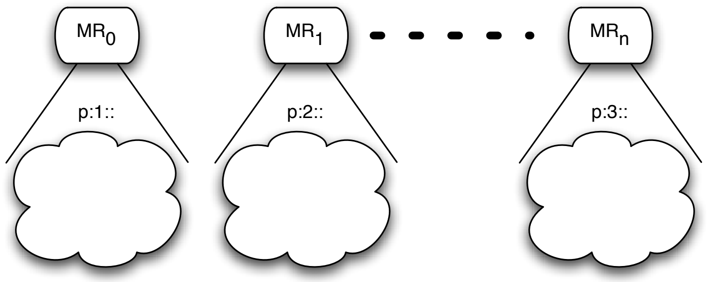


Figure 4.12: MANET routers: which are on the same hierarchical level thus having non-overlapping prefixes.

4.8 Conclusion

This chapter has described goals for and constraints on IP address and prefix autoconfiguration mechanisms for mobile ad hoc networks, which are necessary for a seamless integration of MANETs with the classical IP architecture.

In doing so, it has been illustrated that the unfortunately commonly used terms *"connected MANET"*, *"disconnected MANET"*, and *"stand-alone MANET"* are unsuitable – in particular since these terms do (i) not allow classification of MANETs with different properties and (ii) do not allow this since the terms are not descriptive of relevant properties in MANETs. Indeed, it has been argued that any MANETs can be classified as being *"connected MANETs"*, rendering the terminology and the classification that the terminology aims at providing useless.

Considering the Internet addressing architecture, a terminology has been proposed which, contrary to the terms indicated above, allows to classify MANETs into two categories, describing properties relevant for each category with respect to autoconfiguration of MANETs. This taxonomy aids in simplifying both the description of the goals of autoconfiguration in MANETs, as well as the architectural considerations that apply for how IP addresses and prefixes are assigned and delegated to interfaces and routers.

While the goals for autoconfiguration in MANETs are not that different from those for autoconfiguration in other network types, the characteristics of the MANET interfaces, the requirement that each MANET router shall acquire a prefix and the possible deployment of MANETs as Autonomous MANETs render verbatim application of existing autoconfiguration mechanisms impossible: the fundamental assumptions behind such existing mechanisms are distinctly different from those of MANETs – core examples hereof have been described in this chapter.

The contributions of this chapter have been published in [45, 24] and as contribution to RFC5889 [42].

Chapter 5

Yet Another Autoconf Proposal (YAAP) for MANETs

Chapter 4 has provided an analysis of the IP architecture of the Internet, as well as a comparison to the architecture of MANETs. A new MANET architecture has been proposed, allowing a seamless integration of MANETs in the Internet: two main goals of MANET autoconfiguration should be fulfilled by any autoconfiguration mechanism. First, every MANET interface needs to be configured with a unique prefix either by negotiation between MANET routers or by delegation of a superordinate prefix. This allocation must respect the specific link-model of a MANET node, meaning that a MANET node is a subnet itself, and thus an assigned prefix of a MANET interface cannot be in the same subnet as any other MANET router. The second goal is to delegate topologically correct prefixes from a border router which connects the MANET to any other network(s) as for example the Internet.

The challenges addressed in this chapter are, to provide such a MANET autoconfiguration mechanism, which (i) provides MANET interface addresses according to the considerations in chapter 4, and (ii) enables prefix-delegation to MANET routers, such that a MANET router so desiring can configure networks and hosts, connected to it via “classic IP links”.

5.1 Chapter Outline

Section 5.2 discusses autoconfiguration mechanisms, proposed in literature, for MANETs. Section 5.3 proposes a novel autoconfiguration mechanism, according to the architecture model presented in chapter 4. Section 5.4 discusses extensions and optimizations possible for this mechanism. Section 5.6 presents a formal verification of the proposed mechanism, by way of model checking, and section 5.7 presents a performance study of the mechanism. This chapter is concluded in section 5.8.

5.2 Related Work

Literature abound with MANET autoconfiguration mechanisms, both as academic publications and proposals to standardization bodies – none of which, however, as of 2011 has become IETF Working Group draft, and therefore candidate for standardization[1]. [87] provides a comprehensive survey of a large number of such mechanisms, classifying them according to their applicability domain (autonomous MANETs vs. subordinate MANET) and capability to handle network partitioning and merger. Common for all these proposed mechanisms is, however, that they do not adhere to the considerations described in chapter 4. Notably, none of the presented algorithms configures /128 or /32 prefixes to the MANET interfaces; rather, they assume the MANET to be a single subnet. Moreover, they do not consider the separation of routers and hosts, and therefore do not provide prefixes to the routers, which may then be used by attached hosts or networks to that MANET router. Some of the presented algorithms depend on specific MANET routing protocols to be in operation in the MANET and thus are not generally applicable. While autoconfiguration mechanisms from academic publications, such as [88, 89, 90], present efficient mechanisms to verify uniqueness of addresses by splitting the range of available addresses into parts, none of these mechanism adhere to the considerations described in chapter 4.

The autoconfiguration mechanism, proposed in this chapter, is inspired by Zerouter [91, 92, 93] and IPv6 Stateless Autoconfiguration [21], neither of which are discussed in [87] as these were conceived for use in wired "classic IP" networks and so not directly applicable to MANETs. Zerouter was an early proposal for enabling home users and small companies without dedicated network operation competencies to build arbitrarily complex and large networks without manual interaction. Different approaches were brought forward in the IETF, however none were eventually standardized and there is thus not a single algorithm representing Zerouter. Commonly, the different approaches assume that a border router of the network receives available address space from an ISP and injects that address space into the Zerouter mesh. That address space is available for use throughout the collection of communicating Zerouters, in order to assign addresses to the router interfaces, and to delegate prefixes to the hosts connected to these routers [92] (*e.g.* by using IPv6 stateless autoconfiguration [21]).

5.3 MANET Autoconfiguration Mechanism

The mechanism proposed in this chapter respects the considerations given in chapter 4, and provides a routing-protocol independent solution to the problem stipulated in chapter 4: configuring unique IP addresses for MANET interfaces and providing unique prefixes to MANET routers. It is inspired by the prefix delegation/construction mechanism of Zerouter (as described in section 5.2), specifically by (i) constructing prefixes such that all addresses/prefixes within the MANET can be aggregated (*e.g.*, for injection into an external routing domain as a single entry), and (ii) by the idea of relying on a router (denoted "*initiating router*") for when a new

[1]Note that a number of individual drafts have been submitted, but no Working Group drafts. Refer to section 1.5 for a description of the IETF standardization process.

router arrives in the network. The signaling and interface address configuration mechanism is based on IPv6 stateless autoconfiguration [21].

It has to be noted that merger and partitioning of MANETs is out of scope of this chapter. The proposed algorithm has to be complemented or extended in a future work in order to guarantee unique prefixes when MANETs are partitioned and merged, *e.g.* by a passive autoconfiguration mechanism such as [94].

5.3.1 Algorithmic Overview

Each MANET router will acquire a prefix, constructed as `d:p:s::` with `d` being a prefix (possibly of size 0), common for the whole site (*e.g.*, a global prefix assigned administratively to a given site, or provided by an Internet gateway), `p` being common for all routers in this MANET, and `s` being unique to a specific router.

The task for a router, appearing in a MANET, can thus be summarized as:

- Acquiring `d` and `p`;
- Selecting `s`, unique within the MANET;
- Configuring own MANET interfaces with addresses from within `d:p:s::` (and with a prefix length of /128 or /32 as appropriate for IPv6 and IPv4, respectively).

A router appearing in a network and wishing to be configured, is denoted a *configuring* router. Through a *Prefix Solicitation* (PS) / *Prefix Advertisement* (PA) message exchange, the router learns of the (already configured) routers in its vicinity, and selects one as *initiator* – the router which will assist in acquiring a valid configuration. The *configuring* router extracts `d` and `p` from the PA received from the selected *initiator*, generates a *tentative prefix* `d:p:s::`[2] and, by way of a Router Solicitation (RS) message requests to its *initiator* that this *tentative prefix* be verified unique within the MANET. The *initiator* then issues and floods an RS, containing the *tentative address* of its *configuring* router, through the MANET. If the *initiator* does not (after due delay and retransmissions) receive a *Router Advertisement* (RA) indicating that the prefix is already in use, it will transmit an *Autoconfiguration Confirmation* (AC) message to the *configuring* router, confirming that the *configuring* router now "owns" `d:p:s::` and can now become a *defensive* router. If the *initiator* receives an RA indicating that the tentative prefix is already in use, it informs the *configuring* router by issuing an RA.

A *defensive* router has two tasks. First, if receiving a RS containing its own `d:p:s::`, it must respond by issuing an RA. Second, if receiving a PS, it must respond by a PA, thus accept becoming *initiator* and act as described above.

The *initiator* and *configuring* routers communicate using link-local multicast to the standardized link-local multicast address for MANET routers (`LL-MANET-Routers` [95]), with the *configuring router* using the *unspecified* address as source. These two routers identify traffic destined to each other by way of Universally Unique Identifiers (UUIDs) [96], embedded in all messages

[2] `s` being generated locally by the *configuring* router, *e.g.*, by a pseudorandom generator.

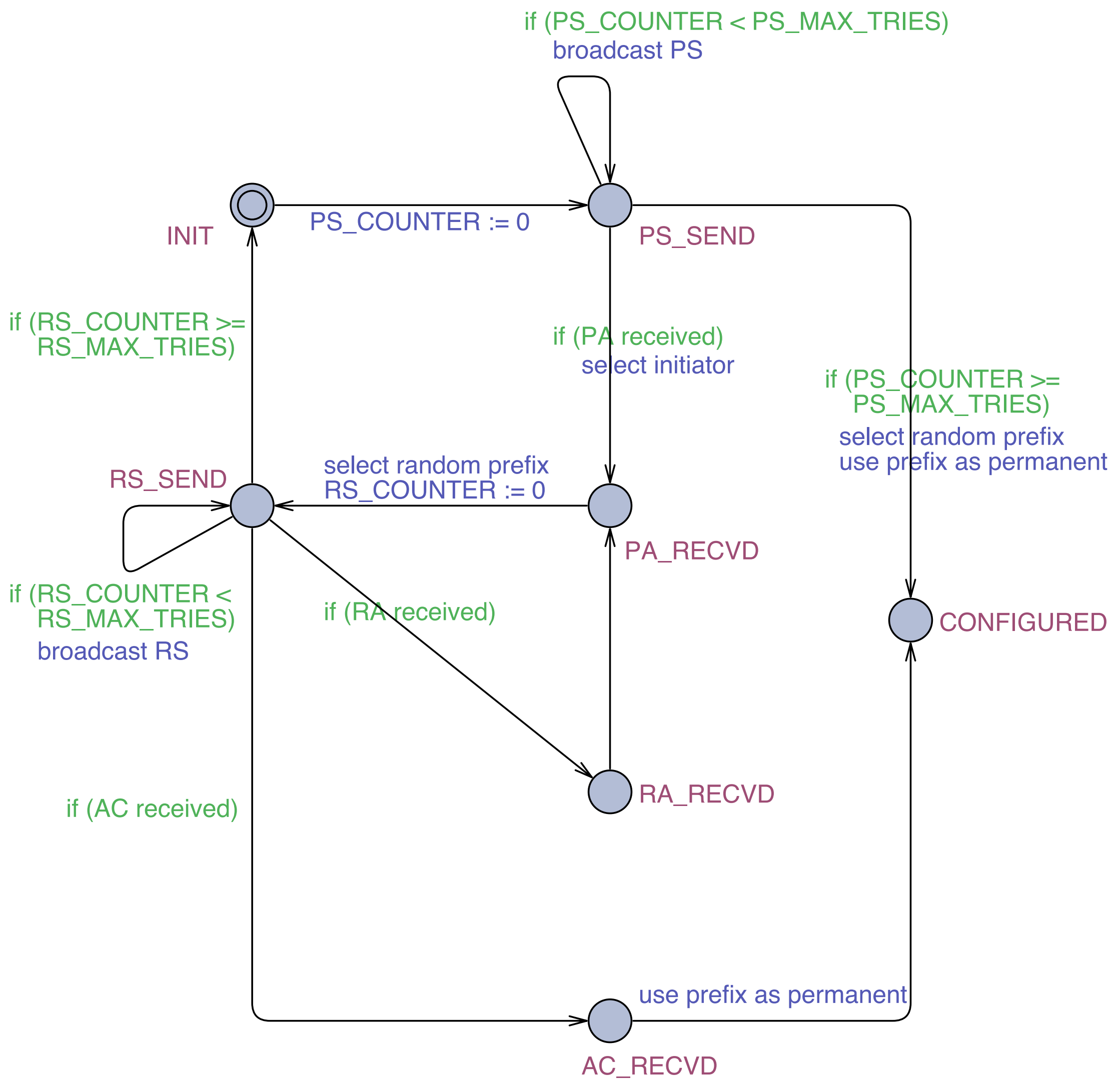


Figure 5.1: State machine of a *configuring* router

exchanged between them. UUIDs are 16 octets long, and as they are exchanged in messages only between neighboring routers, they need only be locally unique. Network-wide messages (RS/RA) are "proxied" by the *initiator*, which is already configured with unique addresses.

5.3.2 Formal Protocol Specification

The protocol is formally specified by way of timed automata, included in this section. The motivation for a formal specification is to prove (or disprove) certain properties, presented in section 5.6.

Figure 5.1 shows the timed automaton representing a *configuring* router. At the initial state, `INIT`, it starts the PS/PA exchange for acquiring the prefix `d:p::`. If, after several retries,

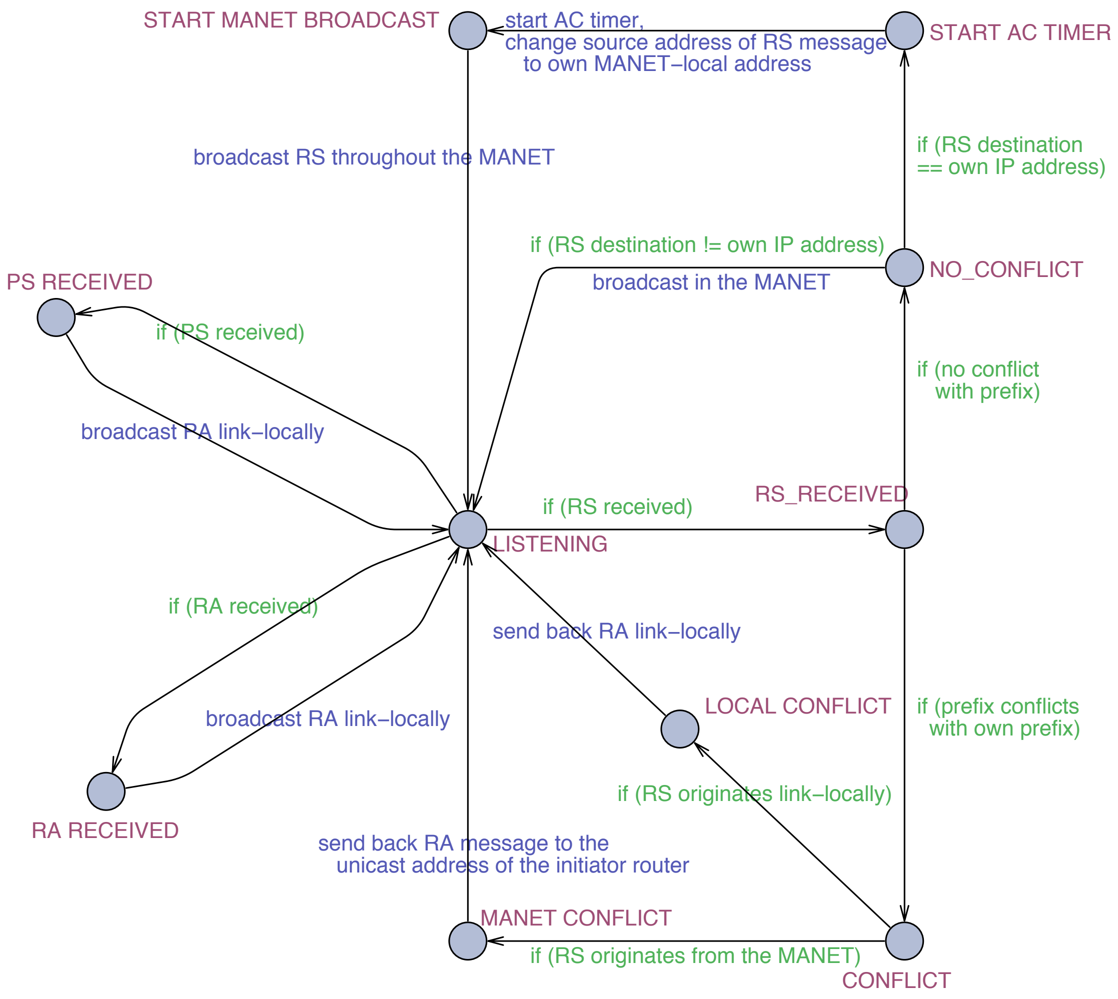

START MANET BROADCAST
start AC timer,
change source address of RS message
to own MANET–local address
START AC TIMER
broadcast RS throughout the MANET
if (RS destination
== own IP address)
if (RS destination != own IP address)
broadcast in the MANET
NO_CONFLICT
PS RECEIVED
if (PS received)
if (no conflict
with prefix)
broadcast PA link–locally
RS_RECEIVED
if (RS received)
LISTENING
if (RA received)
send back RA link–locally
LOCAL CONFLICT
if (prefix conflicts
with own prefix)
broadcast RA link–locally
if (RS originates link–locally)
RA RECEIVED
send back RA message to the
unicast address of the initiator router
MANET CONFLICT
if (RS originates from the MANET)
CONFLICT

Figure 5.2: State machine of a *defensive* router

no PA has been received, the router selects a random `s` and finishes configuration in the state `CONFIGURED`. If, however, it receives a PA from the *initiator* (`PA_RECVD`), it starts sending RS'es to the *initiator* (`RS_SEND`), until it receives an RA (`RA_RECVD`) in case of a conflict, or an AC otherwise (`AC_RECVD`). It then finishes configuration (`CONFIGURED`), becoming a *defensive* router.

Figure 5.2 shows the timed automaton representing a *defensive* router, initiated in the state `LISTENING`. When the router receives an RS, it checks for a conflict with its own prefix (`CONFLICT`) – if one is identified, it responds by sending an RA, either via a link-local transmission if the *defensive* router is the *initiator* for the *configuring* router (`LOCAL_CONFLICT`), or through the MANET otherwise (`MANET_CONFLICT`). If the tentative prefix does not conflict with its own prefix (`NO_CONFLICT`), the defensive router either starts the AC timer, if it is the *initiator* for the *configuring* router (`START_AC_TIMER`), or otherwise forwards the RS through the MANET (back to `LISTENING`).

5.3.3 Autoconfiguration Example

Figure 5.3(a) displays the message exchange for an autoconfiguration example, in which there are no conflicts; the *configuring router* proposes a *tentative prefix*, which happens to be unique in the MANET and, thus, by the end of the exchange, the *configuring router* becomes a *defensive* router. Figure 5.3(b) depicts an autoconfiguration example, in which some other router already is using the *tentative prefix*, and thus where an RA is returned to the *configuring router*.

5.4 Extensions and Optimizations

The protocol proposed in section 5.3 is able to assign unique interface addresses and prefixes to MANET routers, without any prerequisites such as an existing routing protocol running in the MANET, or specific properties of link-local addresses. In this section, optimizations and extensions are proposed, for the purpose of increasing the efficiency of the protocol – especially if another protocol should be running and maintaining useful state. The mantra is: the autoconfiguration protocol should be able to function in isolation – but would be stupid to not take advantage of information from other protocols, if available. Note that in the performance evaluation of the protocol and extensions thereof in section 5.7, only the extension presented in section 5.4.1 is compared with the base algorithm.

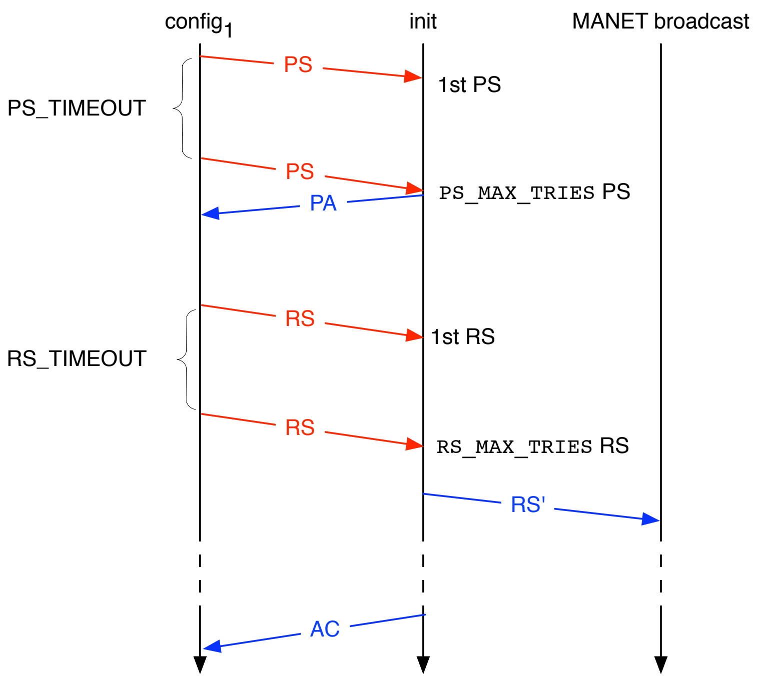
5.4.1 Proxying

Rather than broadcasting RS'es through the MANET, an intermediate router can reply with an RA on behalf of a conflicting router if that intermediate router already knows that there will be a conflict. This may be if the intermediate router already has a route recorded to the prefix contained in the RS (*e.g.*, as provided by a proactive routing protocol such as OLSR [79]), or by an intermediate router temporarily caching RS and RAs previously seen. This may reduce network congestion on a large scale, quantified in section 5.7.

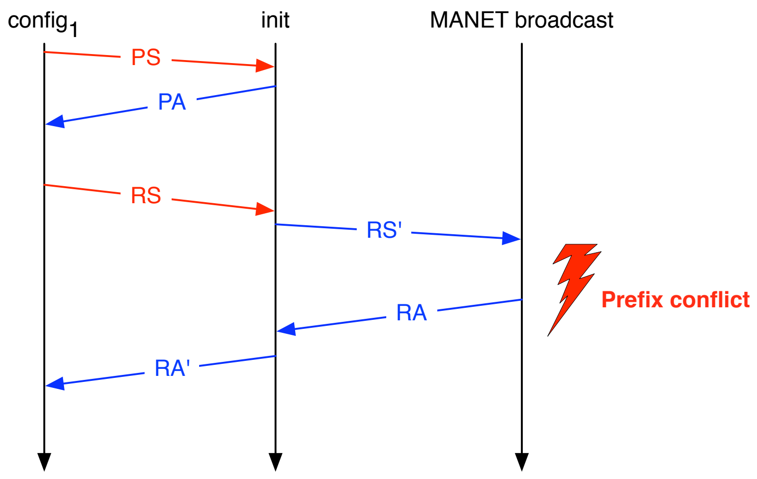
Proxying has also some disadvantages: In particular, many prefixes may be listed as already in use but would actually be available as the routers that have had that prefix have disappeared or changed their prefix. This may lead to a longer time needed for autoconfiguration as new routers may have to test several prefixes with RS messages.

5.4.2 Unicast RAs

In the protocol as presented, the RA from a conflicting router is flooded throughout the MANET in order to be independent from the existence of any routing protocol. If a unicast route is available from the conflicting router to the *initiator*, *e.g.*, as provided by a proactive routing protocol such as OLSR [79], then this route can be used to transit the RA, avoiding the overhead of a flooding operation. Absent a routing protocol, an intermediate router forwarding an RS may install a temporary route towards the source of the RS (the *initiator*), similar to the approach taken in reactive protocols such as [97]. The performance hereof is quantified in section 5.7.1.



(a) Autoconfiguration example without conflict



(b) Autoconfiguration example with conflict

Figure 5.3: Autoconfiguration proceeding example as time diagram

5.4.3 Optimized Broadcasting

RS's are not directly broadcast *configuring* router itself, but are sent to its *initiator* which generates and broadcast the RS on its behalf. Thus, if the MANET supports optimized broadcast, such as MPR flooding as presented in section 6.3.2, this may be exploited to reduce the protocol overhead from RS flooding.

5.4.4 Periodical Prefix Propagation

As an extension to the initial autoconfiguration algorithm it is proposed to have all configured routers periodically broadcast PA messages in their 1-hop neighborhood (possibly embedded in control traffic of a routing protocol). Whenever two MANETs are in close proximity, they can use the information gathered from the broadcast PA messages and join the other MANET for merging and partitioning of MANETs.

5.4.5 IPv4 vs. IPv6

The proposed algorithm works both for IPv4 and for IPv6, since it does not rely on unique link-local addresses, but uses UUIDs for discerning routers, which are of sufficient length to make collisions in the local neighborhood very improbable.

5.4.6 Initiator Selection

In the base algorithm, the *configuring* router has to select its *initiator* if several PA messages arrive from different MANET routers. A trivial choice would be to select the first PA message that arrived. However, this may not be the best choice. The *initiator* routers answering the PS request may be located in different MANETs and thus offering different services. So it may make sense to include information about the connected MANET in the PA message. Examples of such information may be (i) the number of routers in the MANET, (ii) the density of the MANET, (iii) the distance to an Internet gateway, or (iv) the distance to a certain address prefix.

5.5 Additional Considerations

This section addresses several special issues that may occur in the protocol and proposes ways how to solve these problems.

5.5.1 Duplicate UUIDs

A question may arise as to the behavior of the protocol in case two MANET routers select the same UUID. While this might seem quite improbable, given that a UUID is 16 bytes long, the ability to generate a truly random UUID depends on the quality of the random number generator available in the router. Typically, routers use a pseudo-random generator algorithm based on a seed and some environmental settings. Thus, when two routers initiate the random generator with the same seed, they may create the same sequence of random numbers. If that is the case, duplicate UUIDs appear.

Consider the example depicted in figure 5.4 with two *configuring* routers and one *initiator* router.

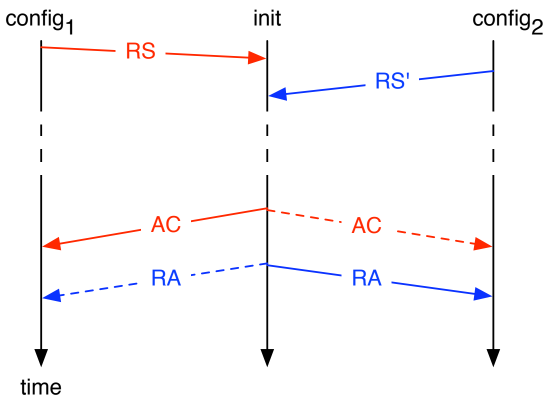


Figure 5.4: Two routers $config_1$ and $config_2$ having the same UUID lead to ambiguous destinations of ACs

The only case when having duplicate UUIDs interferes with the autoconfiguration protocol is when the *initiator* router sends an AC to one of its *configuring* routers, and there happen to be two *configuring* routers in radio range of the *initiator*, and both of these *configuring* routers want to configure the same *tentative prefix*. In the example, both routers $config_1$ and $config_2$ send their RS requesting the *initiator* to verify uniqueness of the same *tentative prefix*, at close to the same time. The *initiator* replies by sending an AC to $config_1$ (assuming there is no prefix conflict in the MANET, of course) and then an RA to $config_2$, with the intent of informing $config_2$ that its suggested *tentative prefix* already is in use. As both of the two *configuring* routers have the same UUID and have requested the same *tentative prefix*, they will both receive the AC – and will therefore both configure with that duplicate prefix. The later arriving RA will be ignored by both routers; they already are configured and thus no longer listening for such.

A simple solution for this problem is depicted in figure 5.5, where the MAC layer and the application layer of each two *configuring* routers are depicted. There are two possible reasons why two routers may have chosen the same UUID:

1. **The two routers have differently seeded random number generators, but simply by "bad chance" generated the same random number.**
   Both routers will probably send the first RS at slightly different times, possibly driven by them contending for access to the MAC layer, say at t_0 and t_0' (according to the routers' local clock) respectively. By including the time of transmission in the RS, and by having the *initiator* store this information, the *initiator* may include this timestamp when it either sends an AC or RA. Both *configuring* routers receiving the RA or AC will compare the included timestamp with the timestamp recorded when sending the corresponding RS. If these timestamps differ, the autoconfiguration process is aborted, and the random number

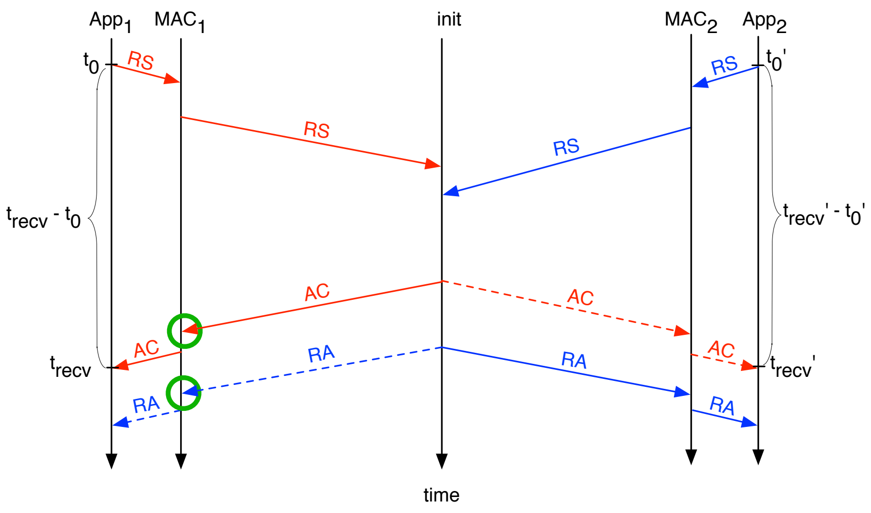


Figure 5.5: Two routers $config_1$ and $config_2$ solving a UUID conflict

generator is seeded with the value $t_{recv} - t_0$ where t_{recv} is the arrival time of the RA or AC. The intent is to create different seeds on both routers as the arriving time of the AC or RA will be probably different at both routers due to the different position and wireless environment. Now that both routers have different UUIDs, the autoconfiguration process can be restarted normally.

2. **The two routers have seeded their random number generator equally**.
   This case is more likely and can easily happen when replicated harddisk images are used for several devices (and seeding the random number generator only with the local time and no additional value like the MAC address). One might think that the RS requests will collide anyway if sent exactly at the same time. But as depicted in the example in figure 5.5, even when the application layer sends the message at the same time t_0, the MAC layer may delay the process of the sending the packet (*e.g.* due to a collision detection mechanism of the MAC layer).

   The objective is to detect a UUID conflict, when it occurs, and once detected avoid entering a CONFIGURED state. To that end, after a router having received an AC or an RA corresponding to its UUID, it will wait a small amount of time. If during that time the router receives a second AC or RA for the same UUID and tentative prefix (depicted as circles in figure 5.5), the router can deduct that a UUID conflict has occurred and take corrective action: re-seeding its random number generator and restarting the autoconfiguration process.

In summary, UUID conflicts are extremely rare. If they occur, however, it is possible to detect the duplicate UUIDs and to resolve the conflict.

5.5.2 No Initiator Router

This problem occurs when two or more routers want to configure a prefix almost simultaneously when there is not yet an already configured MANET router. In figure 5.6, an example of that case is depicted. Both routers $config_1$ and $config_2$ start sending their PS almost simultaneously. As there is no configured router yet, no PA replies will arrive. After `PS_MAX_TRIES` tries, both routers will become *initiator* routers but not being part of the same MANET (*i.e.* not having the same prefix part `p` and not having verified uniqueness of their prefixes).

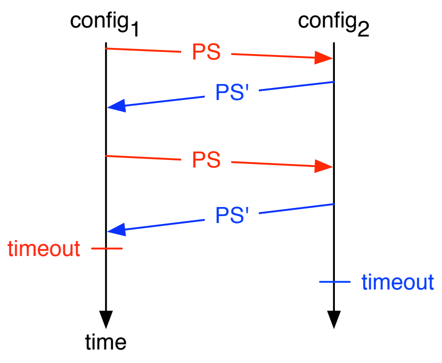


Figure 5.6: Problem when several routers are configured almost simultaneously without *initiator* routers

There can be two different cases as soon as the *configuring* routers from the example become *initiators* and choose the random prefix `p:s`:

1. If the routers choose different prefix parts `p`, they cannot be aggregated using Classless Inter-Domain Routing (CIDR) [98]. They may join the same prefix region by merging.

2. If both routers choose the same prefix part `p`, they are part of the same prefix region but did not verify the uniqueness of the prefix part `s`. They could for example passively detect collisions (as proposed by a number of "passive" autoconfiguration algorithms in the survey [87]).

To solve this problem, *configuring* routers can listen for incoming PS messages from other routers. The router with the lower UUID waits at least `AC_TIMEOUT` seconds before continuing the autoconfiguration process. Thus, the adjacent router can get configured in the meantime.

5.5.3 Initiator Proxying

The following explanation is based on the example depicted in figure 5.7(a). Assuming four routers of which two are already configured ($init_1$ and $init_2$) and two other routers want to configure prefixes ($config_1$ and $config_2$). $config_1$ and $init_1$ are adjacent and $config_2$ and $init_2$ as well. $init_1$ and $init_2$ are connected through the MANET (*i.e.* they are not compulsorily adjacent neighbors – there may be one or several routers between them).

After the exchange of PS and PA messages, both routers $config_1$ and $config_2$ choose the same prefix as tentative prefix which is assumed to be different from the prefixes of $init_1$ and $init_2$. They then broadcast their RS messages to their *initiators*. Both *initiators* broadcast the request throughout the MANET and activate the AC timer. In the meantime, each of them receives the RS requests for the same prefix from the other *initiator* router. But as in original algorithm of section 5.3 the tentative prefix from the RS is only compared with the prefix of a router (and not with the *configuring* routers of this *initiator* router), none of the routers $init_1$ and $init_2$ will reply with an RA. After the timeout, both *initiator* routers will thus send an AC confirming the tentative prefix to their *configuring* routers.

To avoid this allocation of duplicate prefixes, the following is proposed (as depicted in figure 5.7(b)): As soon as the *initiator* router receives the RS request from the *configuring* router (and has no conflict with the *initiator* router itself), an entry in a table is added which includes the tentative prefix of the *configuring* router and its UUID. For this to work the *configuring* router has to include its own UUID in the RS. From the moment of the arrival of the RS from the *configuring* router at the *initiator* router, the *initiator* router serves as a proxy. Whenever a RS request for the same prefix arrives, it checks for conflicts with its own prefix *and* with all entries in the table. The table entry is deleted only after the end of the interval v which is the time from the arrival of the RS to the sending of either an AC or an RA to the *configuring* router. In the example both *initiator* routers will send RA messages back through the MANET when they receive the RS through the MANET as they both have a table entry with the tentative prefix for their *configuring* router.

Note that the same algorithm applies also for the situation of only one *initiator* router which configures two adjacent *configuring* routers almost simultaneously.

5.6 Formal Validation using a Model Checker

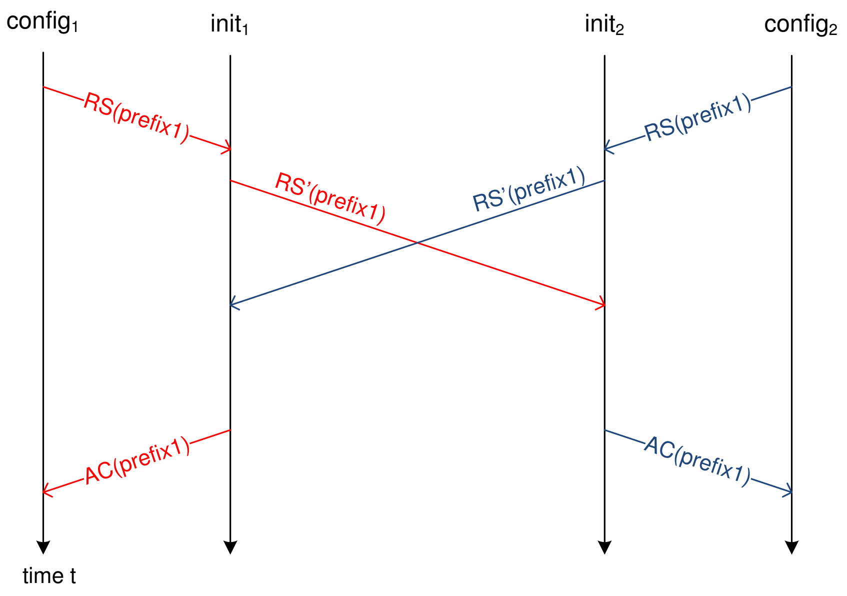
The proposed protocol, as formally specified in section 5.3.2, has been modeled as Timed Automaton and subjected the UPPAAL model checker [99]. This model checker has previously and successfully been applied for other communication protocols (*e.g.* Zeroconf [100]).

5.6.1 Assumptions and Simplifications of the Model

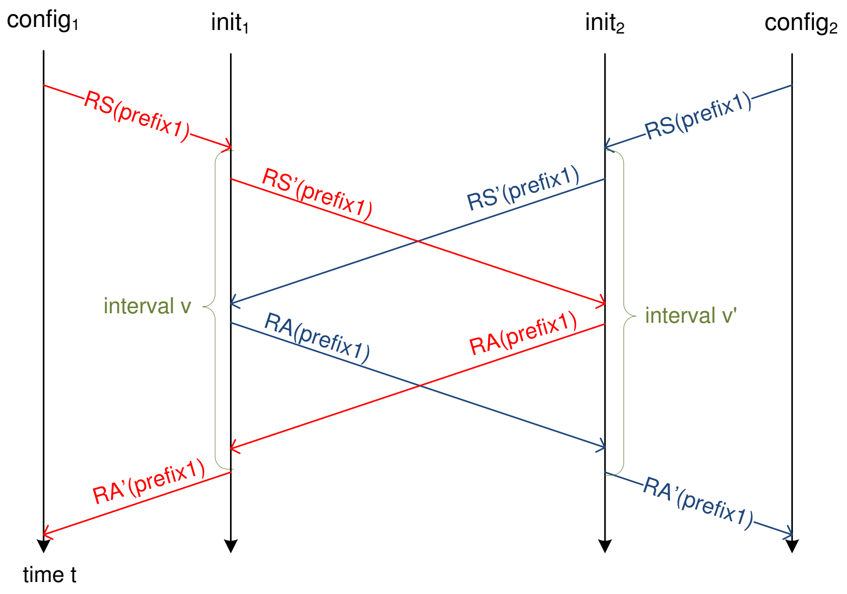
The model used in the model checker has been slightly simplified in comparison to the specification and the implementation. The following assumptions have been made:

- **Simplified broadcasting**
  When the initiator router broadcast an RS, it instantly received by all other configured routers in the MANET, *i.e.*, assuming a perfect and instantaneous broadcast operation.

- **No two routers are configured at the same time**
  If two routers are configured in the same time, with no *initiator* router, no RA or AC exchange can be performed and both routers would become initiator routers without having



(a) Without proxying, two *configuring* routers $config_1$ and $config_2$ could take the same prefix



(b) Proxying in interval v avoids allocation of duplicate prefixes

Figure 5.7: *Initiator* routers serving as proxies for their *configuring* routers

checked address uniqueness. In an initial model of the protocol, this had happened. So using the model checker, a solution for this problem has been developed (as described in section 5.5.2).

- **Routers do not have a MANET prefix**
  Routers all have addresses of the form x where $x \in \mathbb{N}$.

5.6.2 Results of Verification of Properties in UPPAAL

The following properties of the autoconfiguration algorithm were of interest to be proved correct: The algorithm...

- **converges:** The algorithm should terminate in finite time.
- **configures unique prefixes to all MANET interfaces**

In order to prove the correctness of the above-mentioned properties, they have to be "translated" into logic queries supported by UPPAAL (using Computation Tree Logic (CTL)). The following queries have been verified on the model of the autoconfiguration algorithm:

- **A[] not deadlock**
  This query makes sure that in no state (A[]) a deadlock occurs. A state is a deadlock state if there are no outgoing action transitions neither from the state itself or any of its delay successors [101].
- **A[] forall (i : UUIDType) forall (j : UUIDType) IP[i] == IP[j] imply (i == j or IP[i] ==0)**
  This query assures that in all states (A[]), for all routers (j : UUIDType), IP addresses are either not yet configured, or not the same as on any other router in the MANET.
- **E<> forall (i : UUIDType) IP[i] != 0**
  This query checks whether there exists a state (E<>) for every router in the MANET (i : UUIDType), in which that router has configured an IP address. That means that the autoconfiguration algorithm has to facilitate that every router can successfully configure an address.

All three queries have been successfully verified with UPPAAL.

The last query of the above list is weaker than proving that all routers *must* be configured after a finite time:

- A[] E<> forall (i : UUIDType) IP[i] != 0.

UPPAAL does not support to verification of this statement. It is currently unclear whether this behavior of UPPAAL is a deficiency of the particular model checker or generally infeasible due to the state explosion and the huge amount of necessary calculation power.

In summary, it has been proved using the UPPAAL model checker that the proposed protocol works, notably that (i) there are no deadlocks in the distributed protocol, (ii) that all configured routers ultimately end up with distinct addresses and disjoint prefixes and (iii) that given a network with sufficient addresses available (*i.e.*, at least one per router), the protocol will converge to a state where all MANET routers are properly configured.

5.7 Simulation

In this section, the performance of the proposed autoconfiguration mechanism is studied, by way of network simulation using NS2 and using relatively standard scenario parameters (1km^2 square, no mobility, very light traffic and a varying number of routers randomly distributed across the area, simulations averaged over 20 runs per data point, current timestamp used as seed for random number generator). Further simulation parameters are listed in table 5.1. Routers, and so autoconfiguration operations, are started consecutively on each router every second, *e.g.* the router with the ID 17 engages the autoconfiguration process 17.0s after simulation start. Each router acquires the MANET prefix either by receiving a Prefix Advertisement (PA) message or, failing that, by choosing a random prefix if it is the first router in the MANET. All routers will, for the purpose of evaluating worst-case performance, want to configure their prefix `p:s::` to be `0:1::`. Thus, from the second router appearing in a MANET, address collisions will appear. Two different versions of the autoconfiguration algorithm have been tested, the basic version presented in section 5.3 and an extended version with proxying, as described in section 5.4.1.

Table 5.1: NS2 settings

Parameter	**Value**
Area	1000m x 1000m
Simulation duration	800 seconds
Number of routers	20 to 100 (in steps of 20)
Initial topology	Randomly uniformly distributed
Mobility pattern	no mobility
MAC	802.11b
Wireless range	250m
Antenna	Omni-directional antenna
Propagation model	TwoRayGround

5.7.1 Simulation Results

Figure 5.8 depicts a visualization of a simulated network, after completion of the autoconfiguration process. Recalling that the prefixes configured are of the form `d:p:s::`, in this particular scenario, the "site" has a common prefix `fec0::`, (corresponding to `d`) with the autoconfiguration protocol having configured five different MANETs (corresponding to `p`): `fec0:3d51::`, `fec0:ab37::`, `fec0:e2d7::`, `fec0:d723::` and `fec0:7c25::` – within each of which each router has acquired a unique router-specific `s`, *e.g.* `fec0:3d:4::`.

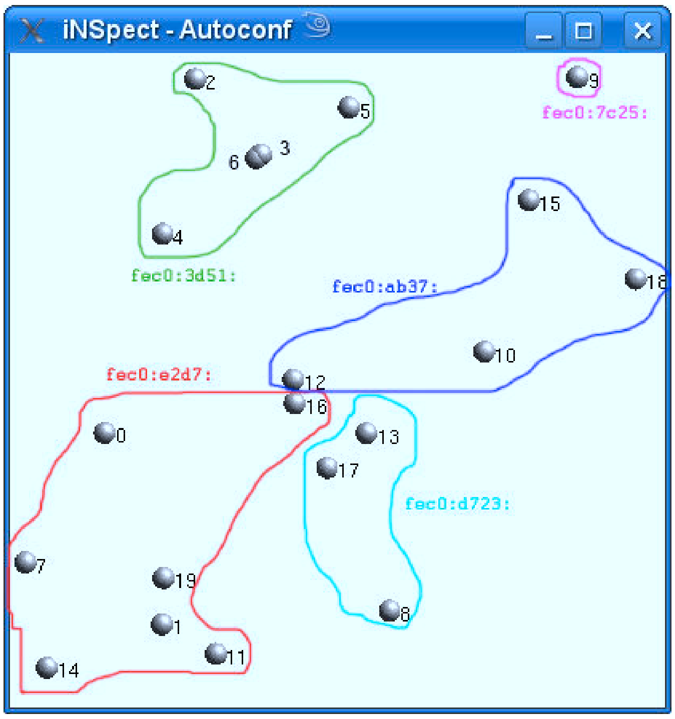


Figure 5.8: MANETs after autoconfiguration

Both versions of the autoconfiguration protocol resulted in all routers being configured with unique prefixes, eventually. However, the number of messages exchanged and number of messages lost due to collisions differ. In the basic version of the algorithm, RS and RA messages are flooded throughout the MANET, whereas in the proxy version, routers cache prefixes extracted from RS messages for the duration of the simulation[3]. Additionally, routers store a temporary reverse route back to the *initiator* when forwarding RS messages, so as to enable unicast RA messages when possible.

Figure 5.9 depicts the total number of transmissions in both versions for the MANET. Note that this number does not include ARP or MAC overhead. As the proposed mechanism does not rely on routing, no IP forwarding is applied. Messages used in this protocol are forwarded on an application layer, which means that they are counted as a new transmission in the figure, each time a message is forwarded or newly generated. As can be seen, proxying significantly reduces the number of transmissions. Consequently, the drop rates, depicted in figure 5.10, are lower due to less channel contention.

[3]In a real-life implementation, the proxy entries should expire after a certain time.

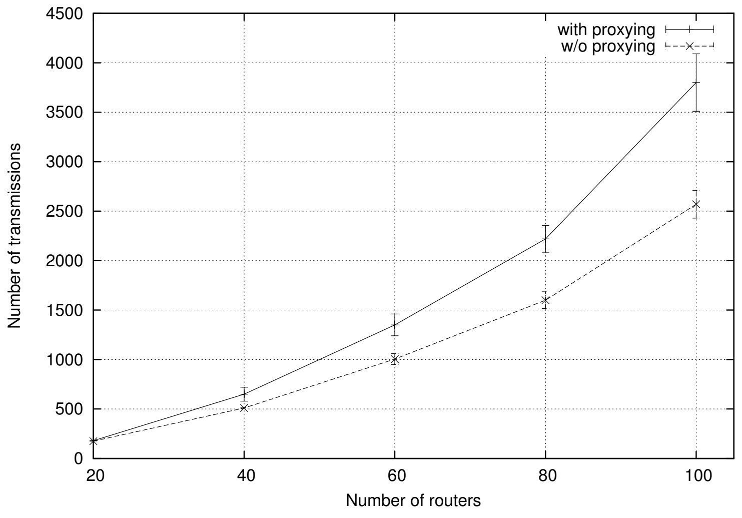


Figure 5.9: Total number of transmissions (90% confidence intervals)

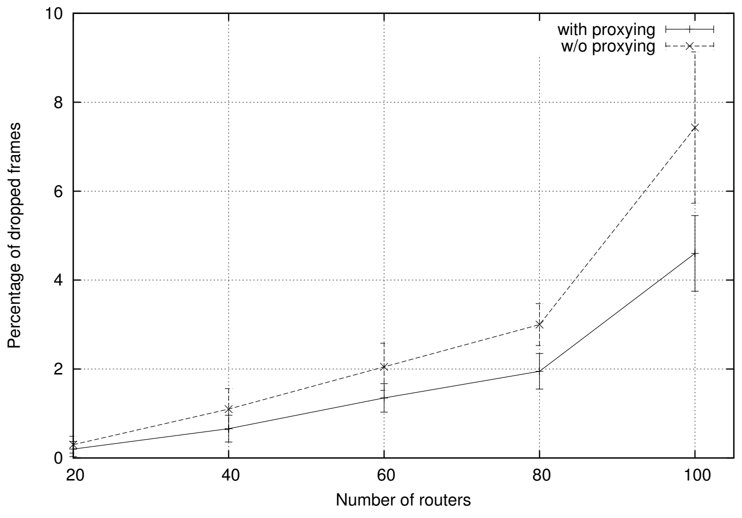


Figure 5.10: MAC frame drop ratio (90% confidence intervals)

5.8 Conclusion

This chapter has presented an autoconfiguration protocol for MANET interface address assignment and prefix delegation to MANET routers, adhering to the architectural considerations presented in chapter 4, by assigning (i) unique addresses and prefixes and (ii) configuring /32 and /128 prefixes respectively. The protocol is inspired by both Zerouter and IPv6 Stateless Autoconfiguration, fully distributed, and enables automatic configuration of both stand-alone MANETs, as well as MANETs connected to an infrastructure providing, *e.g.*, globally scoped addresses/prefixes for use within the MANET. The protocol is specified by way of a set of timed state machines, which has enabled subjecting it to model checking using the UPPAAL model checker – identifying that the protocol works, notably that (i) there are no deadlocks in the distributed protocol, (ii) that all configured routers ultimately end up with distinct addresses and disjoint prefixes and (iii) that given a network with sufficient addresses available (*i.e.*, at least one per router), the protocol will converge to a state where all MANET routers are properly configured.

In order to understand the performance of the protocol, a simulation study has been carried out, testing various performance-improving, overhead-reducing extensions to the protocol, in different scenarios with increasing number of routers in the network. These performance results have shown that the protocol has a relatively low overhead, and that using a proxy extension to the base protocol can further reduce packet overhead and drop rate in the network.

The contributions of this chapter have been published in [24].

Part III

OLSRv2 Performance and Scalability

Chapter 6

Optimized Link State Routing Protocol Version 2 (OLSRv2) Overview

Chapter 4 has proposed an architectural model for ad hoc networks that allows them to be integrated into the Internet and to configure unique and topologically correct IP addresses and prefixes on ad hoc routers, and chapter 5 has proposed such an autoconfiguration mechanism. Once routers are configured with addresses and prefixes, a routing protocol can be deployed on the routers in order to establish end-to-end communication between hosts, as described in chapter 1.

This manuscript proposes in the following chapters several extensions and deployments of the Optimized Link State Routing Protocol version 2 (OLSRv2) [34, 35, 36, 37, 39], specified by the MANET Working Group of the IETF. The objective is to improve stability, performance and scalability of the routing protocol and deployments thereof. This section provides an overview of the basic mechanisms of the protocol.

OLSRv2 retains the same basic algorithms as its predecessor, OLSR [79], however offers various improvements, *e.g.* a modular and flexible architecture allowing extensions, such as for security, to be developed as add-ons to the basic protocol. [102] provides a detailed comparison of the different mechanisms in OLSR and OLSRv2.

6.1 Chapter Outline

OLSRv2 being a link state routing protocol, section 6.2 describes the basic mechanisms of classic link state routing protocols. Section 6.3 then provides an overview over the specifics of OLSRv2.

6.2 Classic Link State Protocol

Each router in a network running a link state routing protocol advertises its list of neighbors throughout the whole network. This information allows each router to perceive the complete topology of the network, and to calculate paths to each destination in the network. In the following, the components of link state routing protocols are described: Neighborhood discovery, link state advertisements, flooding, and shortest path calculation.

6.2.1 Neighborhood Discovery

In order to be able to advertise the list of neighbors of a router throughout the network, routers must have a means to discover adjacent routers, called "neighborhood discovery". One way of discovering neighbors, independent of the underlying link type, is for a router to broadcast "HELLO" control messages advertising the presence of the router. An adjacent router receiving the HELLO deducts that the message originator is a bidirectional neighbor of itself (*i.e.* if a router *a* receives a HELLO from router *b*, it is assumed that *b* can also receive HELLOs from *a*). Figure 6.1 depicts an example neighborhood discovery: router *b* periodically broadcasts HELLO messages, received by router *a*. Router *a* would emit HELLOs as well, which are received by *b*.

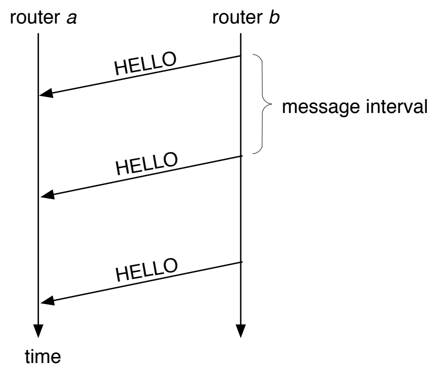


Figure 6.1: Neighborhood discovery

In order to detect updates to the local topology of a router, HELLOs are broadcast periodically with a predefined message interval. Upon reception of a HELLO of a neighbor, a timeout is set for that neighbor. If no further HELLOs are received from that neighbor within the time until the timeout, the neighbor is removed from the neighbor list. Figure 6.2 depicts such a timeout, where due to, *e.g.*, loss of several HELLO messages, router *a* will assume that *b* is no longer a neighbor. Another reason why *a* does not receive HELLOs from *b* any more could be that due to mobility, *b* has "moved away" from *a*.

6.2.2 Link State Advertisements

In link state routing protocols, each router "advertises" (*i.e.* lists) all of its neighbors, acquired by the neighborhood discovery process described in section 6.2.1, in Link State Advertisement

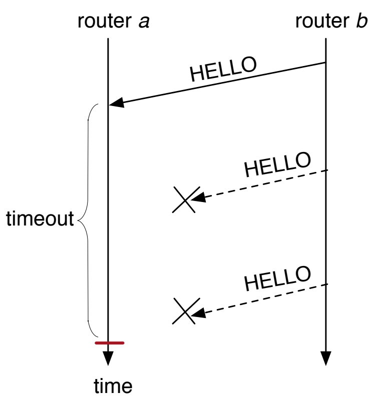


Figure 6.2: Timeouts are used in the neighborhood discovery in order to detect updates to the topology

(LSA) messages which are "flooded" throughout the network. Figure 6.3 shows an example topology of a network. Router *a* has acquired a list of neighbors through the neighborhood discovery process, containing the "gray" routers 1 to 4. The router transmits periodic "LSA" messages containing a list of identifiers of its four neighbors. These messages will be flooded to all routers in the network, including routers 5 to 8. The process of flooding is described in section 6.2.3.

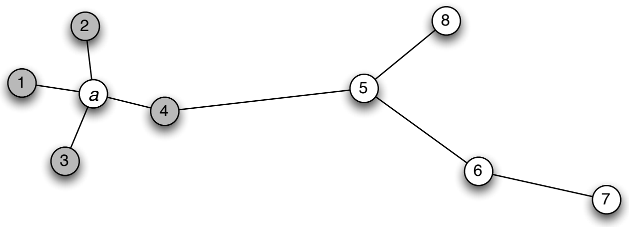


Figure 6.3: Link state advertisement

6.2.3 Classic Flooding (CF)

Flooding is the process of disseminating a message, originated from one router in the network, to all other routers in the network. "Classic flooding" is a basic such flooding mechanism, which is illustrated in figure 6.4. The central router in the middle of the figure transmits a message to be flooded throughout the network. All its direct neighbors will retransmit the message, such that all routers two hop away from the message originator will receive the message (and retransmit it again).

Routers will not unconditionally retransmit received messages in order to avoid "loops", where a router *a* retransmits a message from router *b*, which in turn would receive and retransmit the message from *a*, etc., leading to an infinite cycle of retransmissions, which would consume all

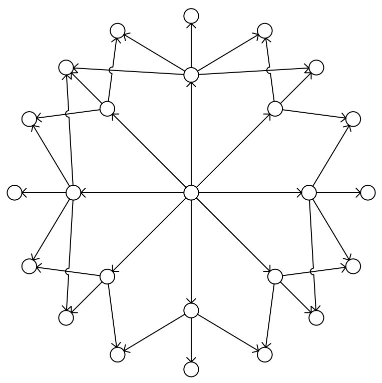


Figure 6.4: Classic flooding

available bandwidth. In order to avoid such redundant retransmissions, one of two mechanisms can be applied:

- sequence numbers: If each message can be identified by a unique sequence number per message originator, routers can store a list of messages (based on these sequence numbers) which they already have forwarded. If the message is received again, it will not be forwarded any more. This requires state on the router to store the list of sequence numbers, and requires that messages contain a sequence number.
- hop limit: Instead of using a sequence number, a hop limit or TTL can be contained in the message, which is decremented at each retransmission. The initial hop limit should be at least as high as the message diameter, in order to allow messages to be disseminated throughout the whole network. Messages with a hop limit of 0 are not forwarded by any router. This mechanism does not completely avoid loops, but does not necessitate state for storing sequence numbers.

6.2.4 Shortest Path Calculation

Once a router has received LSA messages from all other routers in the network, it has a full view of the network topology in form of a graph with vertices being routers and edges being bidirectional links between adjacent routers. An edge also has a “weight”, *i.e.* a cost associated with the link when sending a packet over it. In order to calculate routes to all other routers in the network, paths are calculated (known as “single-source shortest path problem“ [103]). “Shortest” in this context means “with the least cost” according to any kind of distance metric, such as hop count, end-to-end delay, available bandwidth, link quality etc.

One such algorithm to calculate shortest paths is the “Dijkstra” algorithm [104], which consists of the following steps, where the router on which the algorithm is executed is denoted “initial router”:

1. Each router in the graph is assigned an initial distance: 0 for the initial router and *infinity* for all other routers.

2. All routers are marked as "unvisited", the initial router is considered as the "current router".

3. For the current router, the *tentative* distances (from the initial router) to all unvisited neighbors of the current router are calculated. If the current router a has a distance $D(a)$, then the distance to a neighbor b is $D(b) = D(a) + c(a, b)$, where $c(a, b)$ is the cost of the edge between a and b. If the calculated value of $D(b)$ is lower than the previous value, the distance is overwritten, in which case the predecessor along the least cost path is set to $P(b) = a$.

4. The current router is marked as "visited" and its distance is *final* and will not be changed afterwards.

5. If all routers have been visited, the algorithm terminates. Otherwise, the "current router" is set to the unvisited router with the lowest distance, and the algorithm continues with step 3.

The algorithm is illustrated in figure 6.5, where the initial router a calculates routes to all other routers, and the numbers next to the edges depict the edge weight.

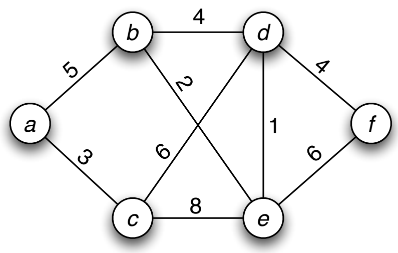


Figure 6.5: Dijkstra algorithm applied for a sample topology

In the initial step of the shortest path calculation (figure 6.6(a) for the example), the distance to the initial router a is set to 0 and to all other routers to *infinity.* Router a is the "current router", and the distances to all unvisited neighbors b and c are calculated ($D(b) = 0 + 5$ and $D(c) = 0+3$). As both the new distance for b and c is lower than the previous (infinite) distances, they are updated with the new values. Router a is marked as "visited" and the algorithm continues with the router with the lowest distance as "current router", *i.e.* router c, depicted in figure 6.6(b). Distances to all unvisited neighbors of c (*i.e.* d and e) are updated to $D(d) = 3+6$ and $D(e) = 3+8$, c is marked as visited and b becomes the current router (figure 6.6(c)). When b updates distances to its unvisited neighbors, the distance to e is reduced from 11 to 7, as the path ($a \rightarrow b \rightarrow e$) is shorter than ($a \rightarrow c \rightarrow e$). The distance to d remains unchanged. The algorithm continues in figures 6.6(d) to 6.6(f), after which all routers have been visited and shortest paths calculated. For example, the shortest path from a to f is ($a \rightarrow b \rightarrow e \rightarrow d \rightarrow f$).

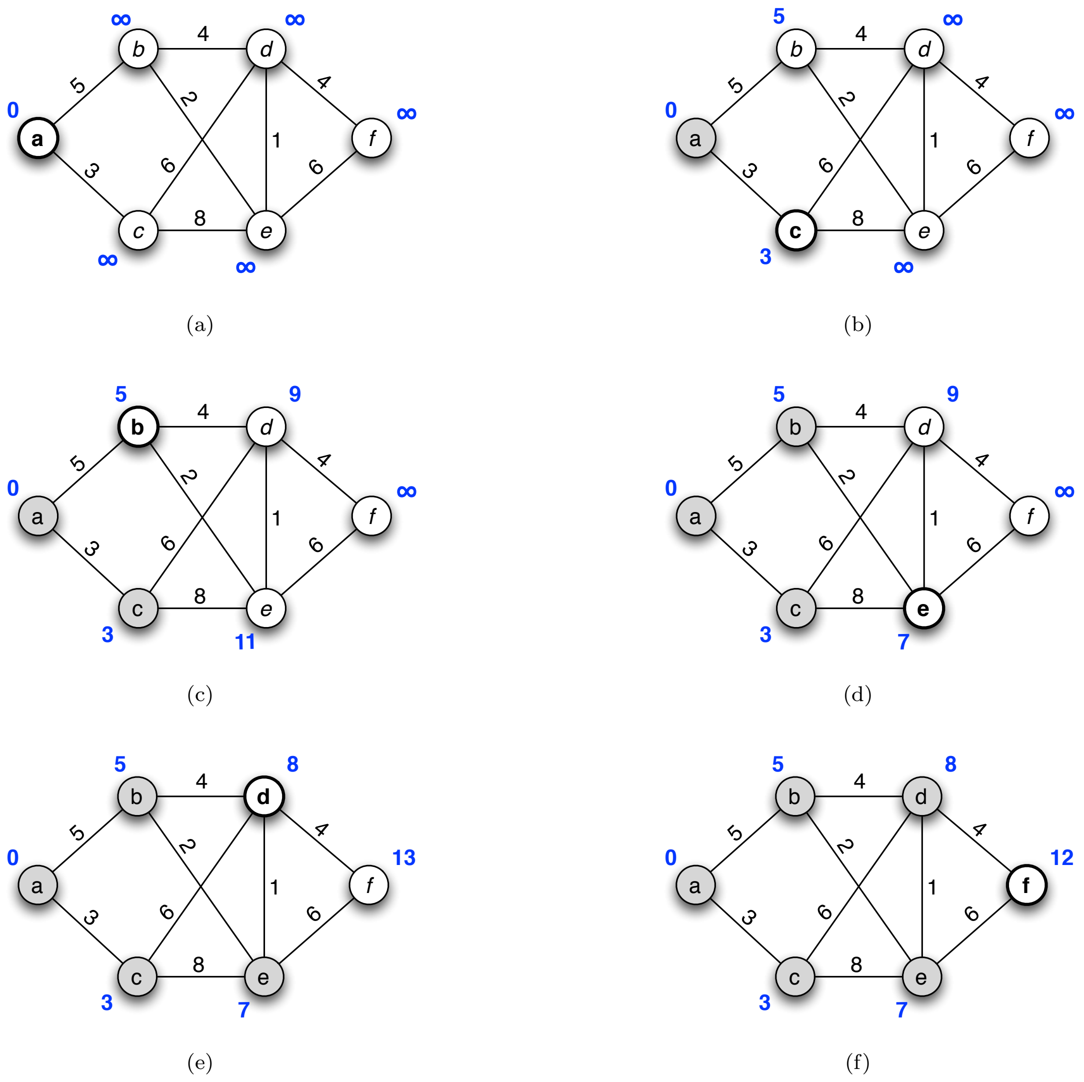


Figure 6.6: Dijkstra algorithm applied to a sample topology. Gray routers are marked as "visited", white routers are "unvisited". The "current router" is written with a bold letter.

6.3 OLSRv2

OLSRv2 retains the same basic mechanisms as classic link state routing protocols, as described in section 6.2. OLSRv2 is, however, optimized for the particularities of ad hoc networks, such as a dynamic topology and asymmetric links, as presented in section 1.2.1. In the following, the core processes of OLSRv2 are presented.

6.3.1 Neighborhood Discovery (NHDP)

Similar to the neighborhood discovery process described in section 6.2.1, each router has to detect its direct neighborhood. NHDP is more complex than the above described neighborhood discovery mechanism, for the following reasons: it (1) provides a bi-directionality test of links to exclude asymmetric links, (2) allows to detect the neighbors of neighbors, *i.e.* "two hop" neighbors, and (3) allows to handle multiple interfaces, each with one or more IP addresses. The reason for providing two hop neighbor detection is further explained in section 6.3.2: OLSRv2 contains an optimized flooding mechanism, which leads to fewer retransmissions and thus lower channel occupation, but which requires topological information of up to two hops distance.

Each router sends HELLOs, listing the identifiers of all the routers from which it has recently received a HELLO, as well as the "status" of the link (HEARD, verified bi-directional – called SYM). A router *a* receiving a HELLO from a neighbor *b* in which *b* indicates to have recently received a HELLO from *a* considers the link *a*-*b* to be bi-directional. As *b* lists identifiers of all its neighbors in its HELLO, *a* learns the "neighbors of its neighbors" (2-hop neighbors) through this process. Figure 6.7 depicts a sample two-hop neighbor detection of three routers *a*, *b* and *c*, where only *b* is in communication range of both *a* and *c*, whereas *a* is not in radio range of *c*. A neighbor that has been previously listed as HEARD or SYM, but from which no HELLO has been received within in a certain time, will be advertised as LOST for a predefined number of HELLOs.

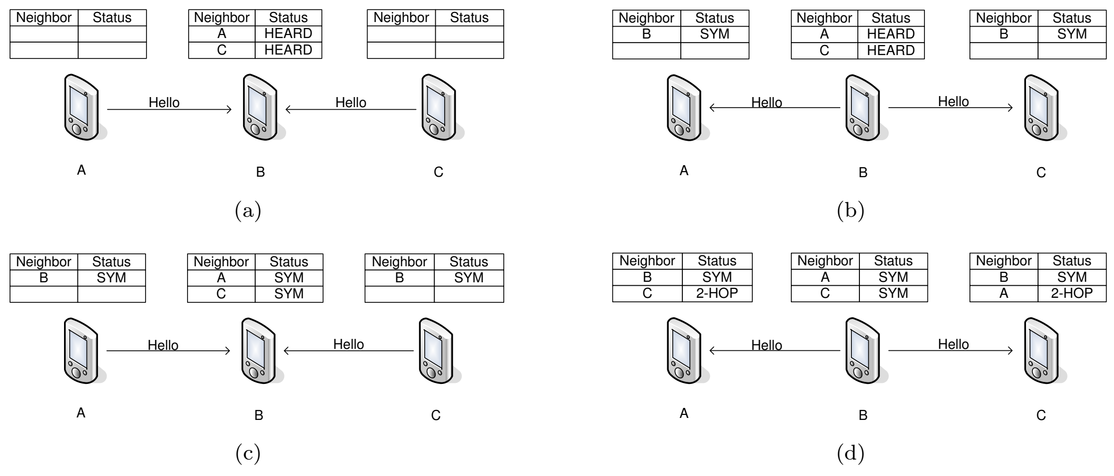


Figure 6.7: Neighbor detection

HELLOs are sent periodically, however certain events may trigger non-periodic HELLOs. NHDP enables each router interface to apply a *link quality mechanism* which, in addition to the message exchange, may constrain when a link is considered as “usable” or not: for example, a router may elect to not consider, and thus not advertise, a link as SYM or HEARD unless a certain ratio of HELLOs are received, unless the Signal-to-noise ratio (SNR) reaches a given threshold etc. Symmetrically, a router may decide to stop advertising a link as SYM or HEARD, subject to similar such constraints.

6.3.2 MPR Flooding (MPRF)

MPR flooding is an optimized variant of the classic flooding process, presented in section 6.2.3. Each router designates, from among its bi-directional neighbors, a subset (MPR set) such that a message transmitted by the router and relayed by the MPR set is received by all its 2-hop neighbors (*i.e.*, the MPR set *“covers”* all 2-hop neighbors). MPR selection is encoded in outgoing HELLOs. The set of routers having selected a given router as MPR is the MPR-selector-set of that router.

Each router may advertise its WILLINGNESS to be selected as MPR in outgoing HELLO messages, as integer number between 0 and a predefined maximum willingness. When selecting MPRs, a router will prefer neighbors with a higher WILLINGNESS and ignore neighbors with WILLINGNESS of 0.

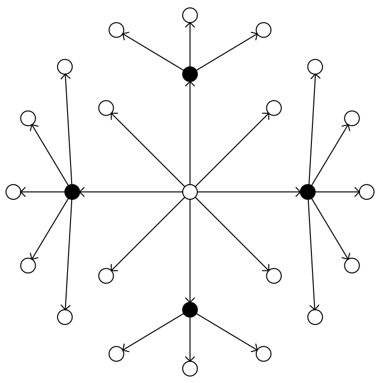
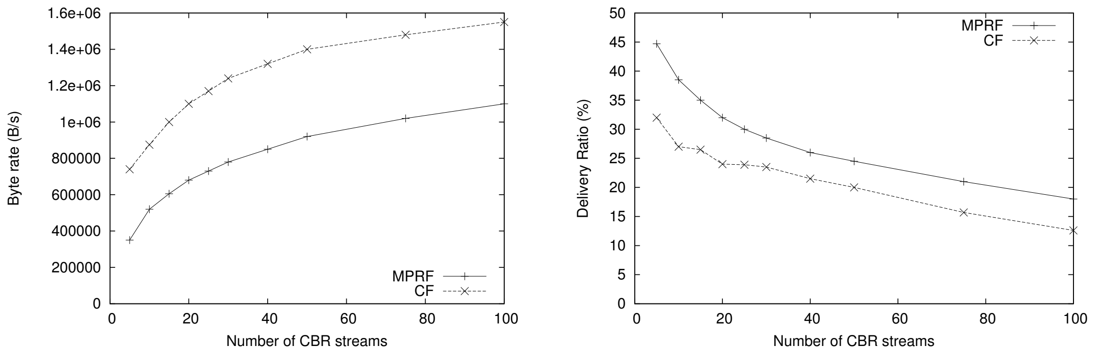


Figure 6.8: Data dissemination using MPR flooding

Figure 6.8 depicts a sample topology with MPR flooding. The central router intends to disseminate a message to all other routers in the network. In classic flooding, each router relays a broadcast packet upon its first receipt by that router. This leads to redundant receptions of the same message by routers two hops away from the central router. Depending on the link layer, collisions and subsequent packet loss may appear. With MPR flooding, only the “black” routers in the example relay the message; leading to fewer redundant retransmissions and packet losses, as well as more available bandwidth for data traffic.

[105] compares different flooding techniques for ad hoc networks, amongst others classic flooding (CF) and MPR flooding (MPRF), and concludes that MPRF leads to lower bandwidth

consumption than CF and that due to fewer collisions the delivery ratio of MPRF is higher than that of CF. Figure 6.9 shows the results of a simulation conducted in [105], comparing delivery ratio and bandwidth consumption, using an NS2 simulation with typical scenario settings (100 nodes, 802.11 interfaces with 250m transmission range, 1400m * 1400m area, 300 seconds simulation time, random waypoint mobility model with node speed between 1 and 2 m/s). Another study of the MPR flooding algorithm can be found in [106].



(a) Bandwidth consumption (b) Delivery ratio

Figure 6.9: Performance comparison of CF and MPRF using an NS2 simulation (data source: [105])

6.3.3 Link State Advertisement

Similar to the LSA process of classic link state routing protocols, presented in section 6.2.2, OLSRv2 routers advertise their neighbors, however, (1) not all routers have to transmit link state advertisements and (2) only a subset of neighbors are advertised in the LSA messages (which are called “Topology Control” (TC) messages in OLSRv2).

Each router, which is elected as MPR by at least one neighbor, must advertise links between itself and its MPR-selector-set, in order to allow all routers to calculate shortest paths. Such link state advertisements, carried in TC messages, are broadcast through the network using the MPR Flooding process. As a router selects MPRs only from among bi-directional neighbors, links advertised in TCs are also bi-directional. TC messages are sent periodically, however certain events may trigger non-periodic TCs. In order to be able to discriminate between fresh and stale information, TC messages, emitted by a given router, include a sequence number incremented each time that router changes the set of links advertised[1].

[1]This TC sequence number is not to be confused with the message sequence number which may be present in any RFC 5444 message, and which serves for avoiding to process and to forward the same message multiple times.

6.3.4 Flexible Message Format

OLSRv2 employs the format specified in [35], for all protocol messages, thereby enabling scope-limited message flooding, compact (aggregated) address representation, also of non-contiguous network addresses, and the ability to associate any number of arbitrary attributes to either of control messages or addresses, by way of inclusion of Type-Length-Value objects (TLVs). The TLV structure permits any given message to be parsed correctly by allowing an implementation to "skip over" TLVs not recognized, thus enabling extensions to be developed that embed information into existing OLSRv2 control messages. The TLV structure is used by OLSRv2, *e.g.* for indicating MPR selection in HELLO messages, or for indicating message emission intervals and the duration for which the content of a message is valid, by way of including TimeTLVs [36].

6.3.5 Inherent Protocol Extensibility

[37, 39] are conceived to enable protocol extensions to be developed for OLSRv2. This is, in addition to the message format described above, accomplished by allowing that subsequent to the usual control message (HELLO and TC) generation, outgoing messages can be handed off to a protocol extension for further processing. Amongst other things, such an extension can insert addresses, Address Block TLVs and Message TLVs. Moreover, upon receipt of a control message, and prior to the usual processing according to that message type, incoming messages can be processed by a protocol extension – including processing of information from that message (extension specific TLVs, for example), as well as allowing a protocol extension to identify the received message as malformed, and thus prohibit processing of that message by OLSRv2.

6.4 Conclusion

The Optimized Link State Routing Protocol version 2 (OLSRv2) retains the same basic mechanisms of classic link state routing protocols: Neighborhood discovery, link state advertisement and flooding of messages throughout the network. However, it is optimized to cope with the specific properties of ad hoc networks, such as asymmetric links and dynamic topology. In order to reduce redundant retransmissions when flooding Link State Advertisements (LSAs) throughout the network, an optimized flooding mechanism ("MPR flooding") is applied in OLSRv2. Moreover, due to the modular design of OLSRv2 and a flexible packet and message format, extensions to OLSRv2 can be easily designed without breaking compatibility of the core specification.

Chapter 7

Java OLSRv2 Implementation

An OLSRv2 implementation, denoted “JOLSRv2”, has been developed in Java, as described in more details in the following subsections. A routing protocol written in Java has several advantages, in particular for research purposes, over the more commonly used C/C++:

- it allows easy prototyping, since it is less error-prone (due to missing pointers, automatic memory management, garbage collection, etc.) and since many auxiliary classes are already contained in the Java distribution (*e.g.* lists, hash tables, tree maps),
- it facilities adding extensions to the core implementation due to class heritage,
- it allows integration of tools as Java applets, also distributed on web pages, and
- it is platform independent.

7.1 Chapter Outline

Section 7.2 presents the architecture of the implementation, such as the different modules of which JOLSRv2 is composed. Section 7.3 presents a GUI client that allows to monitor and control a JOLSRv2 instance using a Java RMI connection. Section 7.4 describes an API that allows for logging of events with different verbosity levels. Section 7.5 presents the operating system dependent routing table API, which is required for adding or modifying routes and which is not feasible with Java code. Section 7.6 details how to add protocol extensions to the existing implementation without the necessity to modify the existing code. Section 7.7 describes some interoperability tools and results from comparing different OLSRv2 implementation with these tools. Section 7.8 concludes the chapter.

7.2 Architecture Overview

According to the separation of the specifications of OLSRv2 into [35], [37], and [39], JOLSRv2 has been implemented as entirely separated Java projects, related as in figure 7.1.

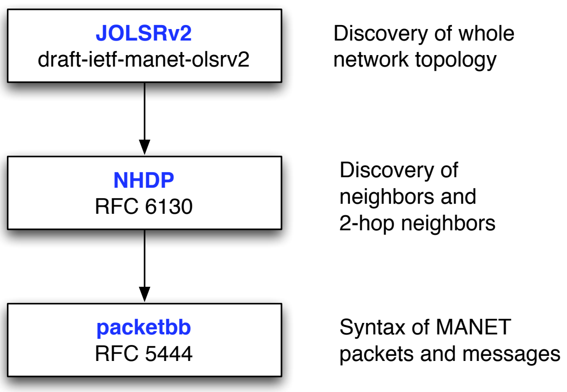

JOLSRv2
draft-ietf-manet-olsrv2
Discovery of whole network topology
NHDP
RFC 6130
Discovery of neighbors and 2-hop neighbors
packetbb
RFC 5444
Syntax of MANET packets and messages

Figure 7.1: Architecture of JOLSRv2

The "packetbb" module (containing the API for using the generalized MANET packet/message format [35]) is a complete, independent library implementation of [35], with an easy-to-use JavaDoc documented API. This is detailed in section 7.2.1. The NHDP implementation relies on the existence of the packetbb library and uses the exposed API hereof. It contains all NHDP entities such as router, interfaces, sender and receiver threads, and is detailed in section 7.2.2. The OLSRv2 module, again, is dependent on the NHDP module and the packetbb library. It uses the existing sets from NHDP, inherits from the NHDP interface and router, and adds all additional functionalities of OLSRv2 (such as sending and processing of TC messages), without rewriting or changing existing NHDP code.

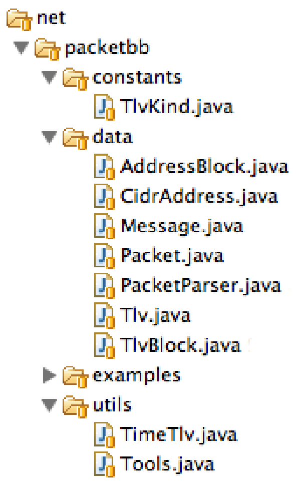
7.2.1 Packetbb

The packetbb library adheres to [35], with all entities from the specification being realized as distinct Java classes (`Message`, `Packet`, `TlvBlock`, `AddressBlock`, `Tlv`). The project is structured as depicted in figure 7.2.

All classes and their class members have been documented using JavaDoc, which allows an easy integration into existing projects. The example in figure 7.3 illustrates how packetbb can be used.

Note that the information within a `Packet` object or `Message` object is not stored in a byte array but in class member variables of the Java class. Thus, before sending a packet to a socket, the method `packet.getBytes()` has to be called which generates the byte array for sending. This enables easy manipulation of a packet or message, at the expense of extra time required for generating the byte array when sending the message.

The packetbb implementation also allows for parsing only parts of a packet, namely the packet header, or packet header plus message headers. This reduces the time spent on parsing messages that are only forwarded (*e.g.* incoming TC messages).



```
net
▼ packetbb
  ▼ constants
      TlvKind.java
  ▼ data
      AddressBlock.java
      CidrAddress.java
      Message.java
      Packet.java
      PacketParser.java
      Tlv.java
      TlvBlock.java
  ▶ examples
  ▼ utils
      TimeTlv.java
      Tools.java
```

Figure 7.2: Structure of the packetbb module

```
Message hello = new Message();
hello.setType(HELLO);
hello.setHopLimit(1);

TlvBlock msgTlvBlock =
    new TlvBlock(MESSAGE);

Tlv valid =
    new Tlv(MSG, VALIDITY_TIME);
valid.setValue(TimeTlv.encode(6,C));
msgTlvBlock.add(valid);

hello.setMsgTlvBlock(msgTlvBlock);
byte[] bytes = hello.getBytes();
```

Figure 7.3: Example: Usage of packetbb

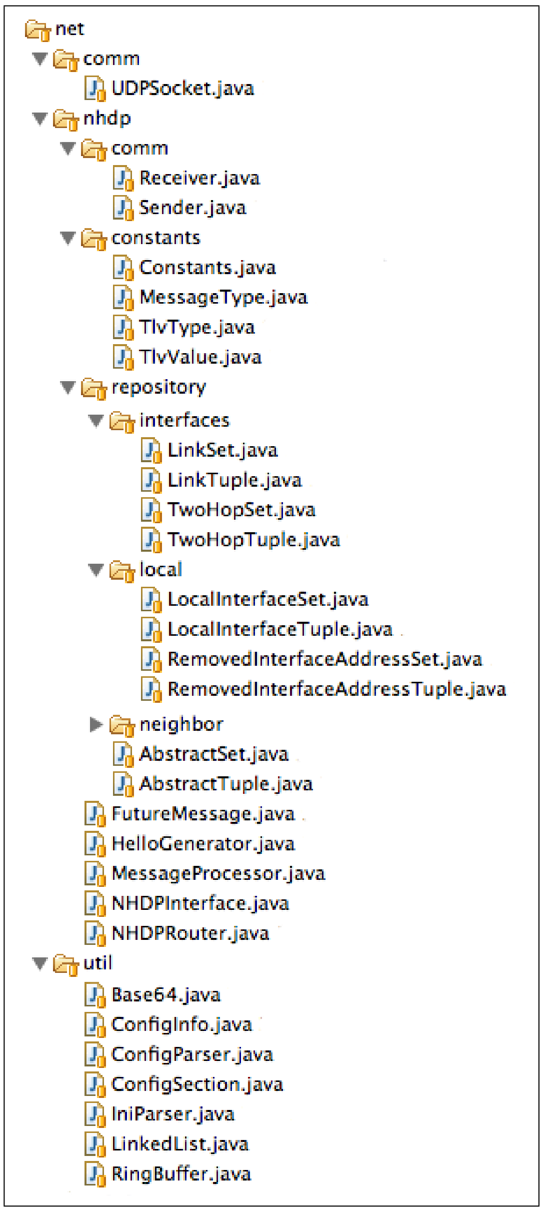
7.2.2 NHDP

The structure of the NHDP implementation is depicted in figure 7.4. It mainly consists of the two classes `NHDPRouter` and `NHDPInterface`, and of classes for all information bases specified in [37]. The classes representing the "sets" and their "tuples" are all derived from a class called `AbstractSet` and `AbstractTuple` respectively. Again, duplicated code for accessing the sets is avoided by using Java inheritance.

Parameters for the router and each of its interfaces can be set from a config file that is read in at startup. For example, the names of the MANET interfaces (*e.g.* wlan0) and their IP addresses can be specified. For changing parameters in a running instance of NHDP (and OLSRv2 respectively), a Remote Method Invocation (RMI) stub has been added. This allows for display of all sets, change of any parameter (such as message intervals), addition or removal of interfaces and IP addresses remotely. A client accessing this information is described in section 7.3.

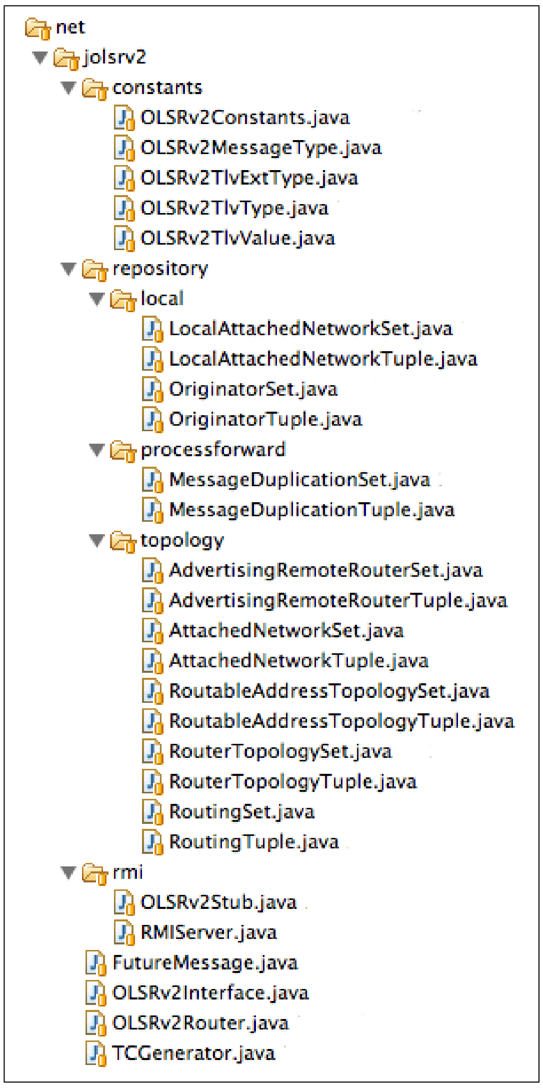
7.2.3 OLSRv2

The OLSRv2 module is a separate module which relies on the NHDP module and the packetbb library. Depicted in figure 7.5, OLSRv2 has two main classes (`OLSRv2Router` and `OLSRv2Interface`) that are each inherited from their NHDP counterparts. Thus, almost no duplicate code needed to be written such as for sending and receiving messages and for generating or processing HELLO messages. The OLSRv2 module basically consists of all additional sets that are specified in [39], all new variables and constants (such as `TC_INTERVAL`), and the generation and processing of TC messages.



```
net
  comm
    UDPSocket.java
  nhdp
    comm
      Receiver.java
      Sender.java
    constants
      Constants.java
      MessageType.java
      TlvType.java
      TlvValue.java
    repository
      interfaces
        LinkSet.java
        LinkTuple.java
        TwoHopSet.java
        TwoHopTuple.java
      local
        LocalInterfaceSet.java
        LocalInterfaceTuple.java
        RemovedInterfaceAddressSet.java
        RemovedInterfaceAddressTuple.java
      neighbor
      AbstractSet.java
      AbstractTuple.java
    FutureMessage.java
    HelloGenerator.java
    MessageProcessor.java
    NHDPInterface.java
    NHDPRouter.java
  util
    Base64.java
    ConfigInfo.java
    ConfigParser.java
    ConfigSection.java
    IniParser.java
    LinkedList.java
    RingBuffer.java
```

Figure 7.4: Structure of the NHDP module

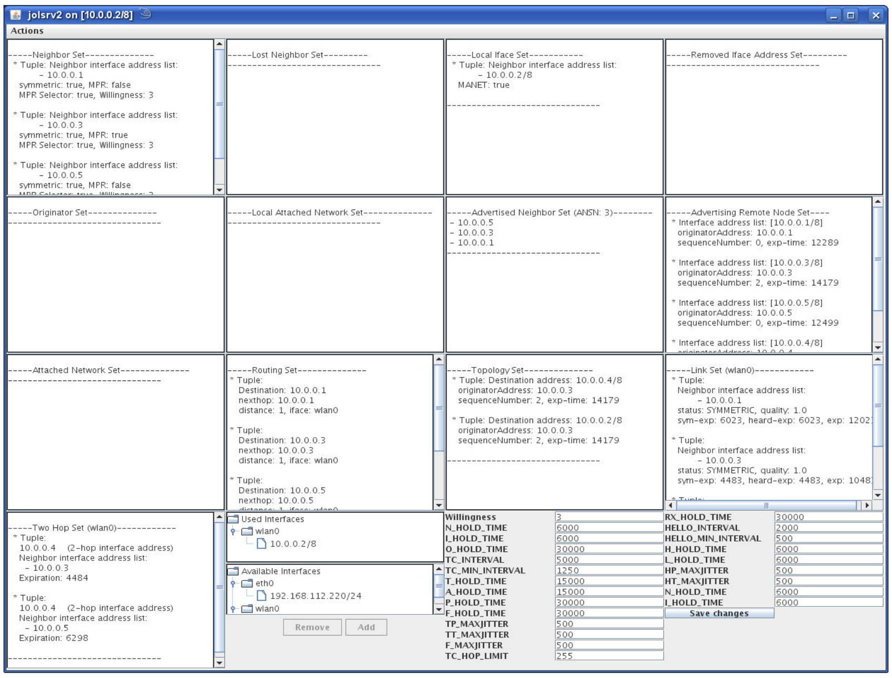


```
net
  jolsrv2
    constants
      OLSRv2Constants.java
      OLSRv2MessageType.java
      OLSRv2TlvExtType.java
      OLSRv2TlvType.java
      OLSRv2TlvValue.java
    repository
      local
        LocalAttachedNetworkSet.java
        LocalAttachedNetworkTuple.java
        OriginatorSet.java
        OriginatorTuple.java
      processforward
        MessageDuplicationSet.java
        MessageDuplicationTuple.java
      topology
        AdvertisingRemoteRouterSet.java
        AdvertisingRemoteRouterTuple.java
        AttachedNetworkSet.java
        AttachedNetworkTuple.java
        RoutableAddressTopologySet.java
        RoutableAddressTopologyTuple.java
        RouterTopologySet.java
        RouterTopologyTuple.java
        RoutingSet.java
        RoutingTuple.java
    rmi
      OLSRv2Stub.java
      RMIServer.java
    FutureMessage.java
    OLSRv2Interface.java
    OLSRv2Router.java
    TCGenerator.java
```

Figure 7.5: Structure of the JOLSRv2 module

7.3 RMI GUI Client

JOLSRv2 contains a Java RMI server, enabling remote access of methods that are interfaced in a stub, with all parameters being serialized by Java. A Java applet has been implemented which allows to connect to the RMI server (using a TCP/IP connection) and to call methods that are defined in the stub. Written in Java Swing/AWT, it regularly acquires all sets from OLSRv2/NHDP, and displays these using their *toString()* methods (depicted in figure 7.6). In addition, all parameters can be changed and new interfaces and IP addresses can be added or removed. This tool can be used for experimental runs of a given test setup – for a deployment, this feature needs to be removed or authenticated for security reasons.



```
-----Neighbor Set--------------
 * Tuple: Neighbor interface address list:
        - 10.0.0.1
  symmetric: true, MPR: false
  MPR Selector: true, Willingness: 3

 * Tuple: Neighbor interface address list:
        - 10.0.0.3
  symmetric: true, MPR: true
  MPR Selector: true, Willingness: 3

 * Tuple: Neighbor interface address list:
        - 10.0.0.5
  symmetric: true, MPR: false
```

```
-----Lost Neighbor Set---------
-------------------------------
```

```
-----Local Iface Set-----------
 * Tuple: Neighbor interface address list:
        - 10.0.0.2/8
  MANET: true

-------------------------------
```

```
-----Removed Iface Address Set---------
-------------------------------
```

```
-----Originator Set--------------
-------------------------------
```

```
-----Local Attached Network Set--------------
-------------------------------
```

```
-----Advertised Neighbor Set (ANSN: 3)--------
 - 10.0.0.5
 - 10.0.0.3
 - 10.0.0.1
-------------------------------
```

```
-----Advertising Remote Node Set----
 * Interface address list: [10.0.0.1/8]
   originatorAddress: 10.0.0.1
   sequenceNumber: 0, exp-time: 12289

 * Interface address list: [10.0.0.3/8]
   originatorAddress: 10.0.0.3
   sequenceNumber: 2, exp-time: 14179

 * Interface address list: [10.0.0.5/8]
   originatorAddress: 10.0.0.5
   sequenceNumber: 0, exp-time: 12499

 * Interface address list: [10.0.0.4/8]
```

```
-----Attached Network Set--------------
-------------------------------
```

```
-----Routing Set--------------
 * Tuple:
   Destination: 10.0.0.1
   nexthop: 10.0.0.1
   distance: 1, iface: wlan0

 * Tuple:
   Destination: 10.0.0.3
   nexthop: 10.0.0.3
   distance: 1, iface: wlan0

 * Tuple:
   Destination: 10.0.0.5
   nexthop: 10.0.0.5
```

```
-----Topology Set--------------
 * Tuple: Destination address: 10.0.0.4/8
   originatorAddress: 10.0.0.3
   sequenceNumber: 2, exp-time: 14179

 * Tuple: Destination address: 10.0.0.2/8
   originatorAddress: 10.0.0.3
   sequenceNumber: 2, exp-time: 14179

-------------------------------
```

```
-----Link Set (wlan0)------------
 * Tuple:
   Neighbor interface address list:
        - 10.0.0.1
   status: SYMMETRIC, quality: 1.0
   sym-exp: 6023, heard-exp: 6023, exp: 1202

 * Tuple:
   Neighbor interface address list:
        - 10.0.0.3
   status: SYMMETRIC, quality: 1.0
   sym-exp: 4483, heard-exp: 4483, exp: 1048
```

```
-----Two Hop Set (wlan0)------------
 * Tuple:
   10.0.0.4    (2-hop interface address)
   Neighbor interface address list:
     - 10.0.0.3
   Expiration: 4484

 * Tuple:
   10.0.0.4    (2-hop interface address)
   Neighbor interface address list:
     - 10.0.0.5
   Expiration: 6298

-------------------------------
```

Willingness	3	RX_HOLD_TIME	30000
N_HOLD_TIME	6000	HELLO_INTERVAL	2000
I_HOLD_TIME	6000	HELLO_MIN_INTERVAL	500
O_HOLD_TIME	30000	H_HOLD_TIME	6000
TC_INTERVAL	5000	L_HOLD_TIME	6000
TC_MIN_INTERVAL	1250	HP_MAXJITTER	500
T_HOLD_TIME	15000	HT_MAXJITTER	500
A_HOLD_TIME	15000	N_HOLD_TIME	6000
P_HOLD_TIME	30000	I_HOLD_TIME	6000
F_HOLD_TIME	30000		
TP_MAXJITTER	500		
TT_MAXJITTER	500		
F_MAXJITTER	500		
TC_HOP_LIMIT	255		

Figure 7.6: RMI client

This applet also allows for displaying the local topology from the point of view of the router to which the applet is connected as depicted in figure 7.7.

7.4 Logging API

JOLSRv2 uses the logging API provided by Java (detailed in [107]). The logging API allows to add one or more "handlers" to the log manager, each of them receiving the log messages and processing them. As per default, there is a `ConsoleHandler`, which outputs the log messages on the standard output of the terminal. Another default handler is the `FileHandler`, which outputs the log messages to a file.

For each of the handlers, a formatter can be defined. The formatter defines how the log

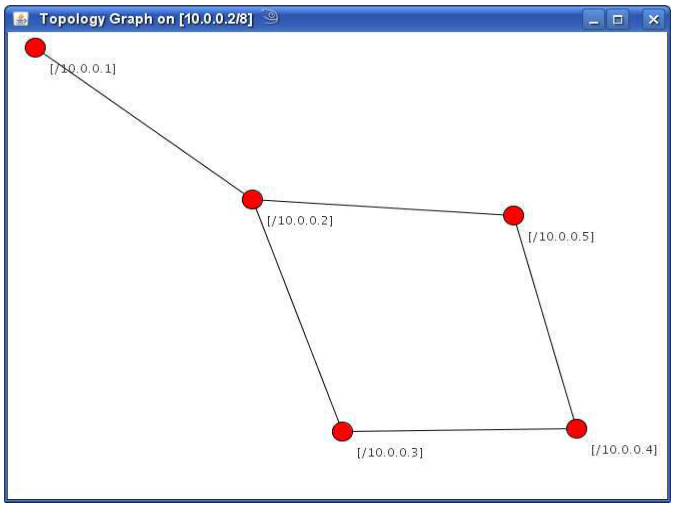


Figure 7.7: Local view of the topology in the GUI applet

message shall be printed out as a string. For example, the `XMLFormatter` outputs each log message in XML code with different information about the log message. Moreover, for each of the handlers, a different logging level can be set, from FINEST to ERROR in increasing order of importance. The handler will only process log messages with that minimum level.

The different settings can be defined in a file called `logging.properties`, which must be located in the same path as JOLSRv2. In the following, a sample logging.properties file is listed:

```
handlers=java.util.logging.FileHandler, java.util.logging.ConsoleHandler

java.util.logging.FileHandler.pattern = jolsrv2.log
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = util.XMLFormatter
java.util.logging.FileHandler.level = FINEST

java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

Note that for logging messages with a level of FINE or lower, the `debug` parameter must be set to `true` in the config.txt file (refer to section C.2 in the appendix C). The reason for this additional switch is to increase performance of JOLSRv2 when no logging is used.

7.5 Routing API for other Operating Systems

While JOLSRv2 runs – without recompilation or modification of the source code – on all operating systems that provide a Java Virtual Machine, the modifications to the kernel routing table are dependent on the Operating System (OS). This code is native to Java (*i.e.* written in C++)

and needs to be provided in a library called `JOLSRv2_RT`. For Mac OS X and Linux, the library is provided in the implementation. For other operating systems (such as Windows), the native methods have to be implemented. This section describes how to implement the necessary code.

In the `jni` subdirectory, the necessary files can be found: `RoutingSet.c` and `RoutingSet.h`, which contain the native parts of the Java `RoutingSet` class. The folder `linux` and `mac` contain the existing native code for Linux and Mac OS X. The folder `empty` contains an empty sample `RoutingSet`, which can be used for implementing the native methods for other operating systems. The methods shown in table 7.1 have to be implemented.

Table 7.1: Native methods in RoutingSet

Native method	**Description**
`int __createEntry(InetAddress destination, int prefixLength, InetAddress gateway, String iface, int metric)`	Creates a new routing entry for the given destination
`int __updateEntry(InetAddress destination, int prefixLength, InetAddress gateway, String iface, int metric)`	Updates an existing route
`int __deleteEntry(InetAddress destination, String iface)`	Deletes a route to a destination
`String __getErrorMessage()`	Returns the error message if an error has occurred in a previous function call
`int __RoutingSetStartup()`	Called from within the constructor of the RoutingSet class, may be used for initialization
`void __RoutingSetShutdown()`	Called during finalization of the RoutingSet class, may be used for clean up

7.6 Writing Extensions

Similar to the way that `OLSRv2Router` inherits from `NHDPRouter` and `OLSRv2Interface` inherits from `NHDPInterface`, extensions can be easily integrated into both OLSRv2 and NHDP. Any method that needs to be changed can be overwritten, for example `generateHello()`. As one possible extension, a signature mechanism for packetbb messages has been implemented that allows messages to be signed and verified on sending and reception respectively (described in chapter 12). This is achieved by inheriting from `OLSRv2Interface` and overwriting `processHelloMsg()` and `generateHello()`. In order to run this new extension, the router and interface class must be defined in the config file that is read on start. For the above-mentioned signature class, the config file would define the parameter `routerclass = net.jolsrv2.SignedOLSRv2Router` and in the interface section `interfaceclass = net.jolsrv2.SignedOLSRv2Interface` and all necessary parameters such as public and private key. The `main()` function of the Java application then invokes the given router and interface class instead of the default OLSRv2Router and OLSRv2Interface classes.

7.7 Interoperability Tests

In order to verify whether the implementation corresponds to the specification, tests with other packetbb, NHDP and OLSRv2 implementations have been conducted at the OLSR Interop Workshops 2008 and 2009 [108]. In the following, the tests are described that have been designed for checking interoperability of different implementations. The results of the comparison are also presented.

7.7.1 Packetbb Tests

Implementations of the packet and message format [35], which is used amongst others by NHDP and OLSRv2, have been tested for correctness. An implementation of a routing protocol using the message format must both correctly *parse* and *generate* packets and messages. A “test” application has been developed in Java, which generates a number of different variations of packets and messages, with the objective to cover most variations of different TLVs, message headers, etc.[1] The packets and messages are sent via UDP to a link-local broadcast or unicast address of a machine running an implementation to be tested. This implementation should parse the data and provide a human-readable output, that can then be manually compared to the generated packets. If the human-readable output is implemented to be the same as the one from the test application, a *diff* [109] can automate the comparison.

The following packets have been constructed for the test (packets are numbered from 1 to 36):

1. empty packet
2. packet 1 with sequence number
3. packet 2 with empty TLV block
4. packet 3 with one Packet TLV
5. packet 4 plus a second TLV with type extension
6. packet 5 with a value set for the TLV
7. packet 6 with the TLV having an extended length
8. packet 7 with an empty (IPv4) message
9. packet 8 plus a second message with an originator address set
10. packet 9 with hop count set for the second message
11. packet 10 with hop limit set for the second message
12. packet 11 with a sequence number for the second message
13. packet 12 with a Message TLV added to the first message
14. packet 13 with the address 0.0.0.0 added to the second message
15. packet 13 with the address 255.255.255.255 added to the second message
16. packet 13 with the address 0.0.0.1 added to the second message
17. packet 13 with the address 10.0.0.0 added to the second message

[1]It is almost impossible to create *all* possible variations of packets and messages due to the sheer number of possibilities.

18. packet 13 with the address 10.0.0.1 added to the second message
19. packet 13 with the addresses 10.0.0.1 and 10.0.0.2 added to the second message
20. packet 13 with the addresses 10.1.1.2 and 10.0.0.2 added to the second message
21. packet 13 with the addresses 10.0.0.0 and 11.0.0.0 added to the second message
22. packet 13 with the addresses 10.0.0.5/16 and 10.0.0.6/24 added to the second message
23. packet 22 with an Address TLV added without index
24. packet 23 with the Address TLV having an index of 1
25. packet 24 with the Address TLV having a start index of 1 and a stop index of 3
26. packet 25 with the Address TLV being a multi-value TLV
27. packet 26 with a second Address TLV with start index 0, stop index 2 and multi-value set
28. packet 27 with an extended length Address TLV
29. packet with an IPv6 message
30. packet 29 with an IPv6 originator address
31. packet 30 with the address 1000::1 added
32. packet 31 with the address 1000::2 added
33. packet 29 with the addresses 1000::2 and 1000::11:2 added
34. packet 29 with the addresses 1000:: and 1100:: added
35. packet 29 with the addresses 1000::5/64 and 1000::6/48 added
36. packet 35 with the same IPv4 message added as in packet 28

The reason for adding different addresses to the messages is to test the address compression mechanism of packetbb. A complete human readable output of the packets as well as the byte sequence of each packet can be found at [110].

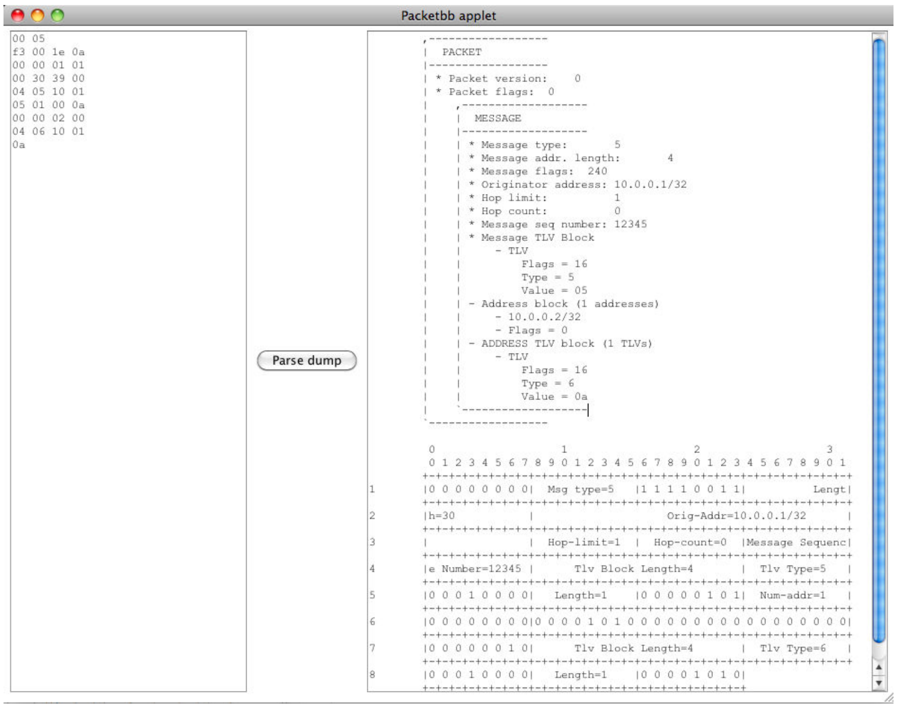
7.7.2 Packetbb Applets

Two applets have been developed, both available at [110], with the objective of providing an easy interoperability test of packetbb implementations. The applet depicted in figure 7.8 allows to "parse" a message, formatted in hex numbers, and to generate a human-readable output as well as a structural view of the packet, as often used in IETF documents.

The applet depicted in figure 7.9 enables to "create" a packet with a graphical user interface, *i.e.* to set header values, add messages, TLV, addresses etc. The packet can then be generated and the resulting byte sequence displayed, together with a human-readable output.

7.7.3 Emulator for large Topologies

In order to test the correct functionality of NHDP and OLSRv2, as well as to measure the scalability of the implementation, an OLSRv2 topology emulator has been built. The basic idea is illustrated in figure 7.10. Note that the emulator runs on *real* machines, *i.e.* the addresses depicted in the figure are bound to a network interface in the operating system. In figure 7.10, a router is depicted with the address 10.0.0.1. This is the router running the OLSRv2 implementation to be evaluated. On the right side, several routers are shown with, in this example, IP

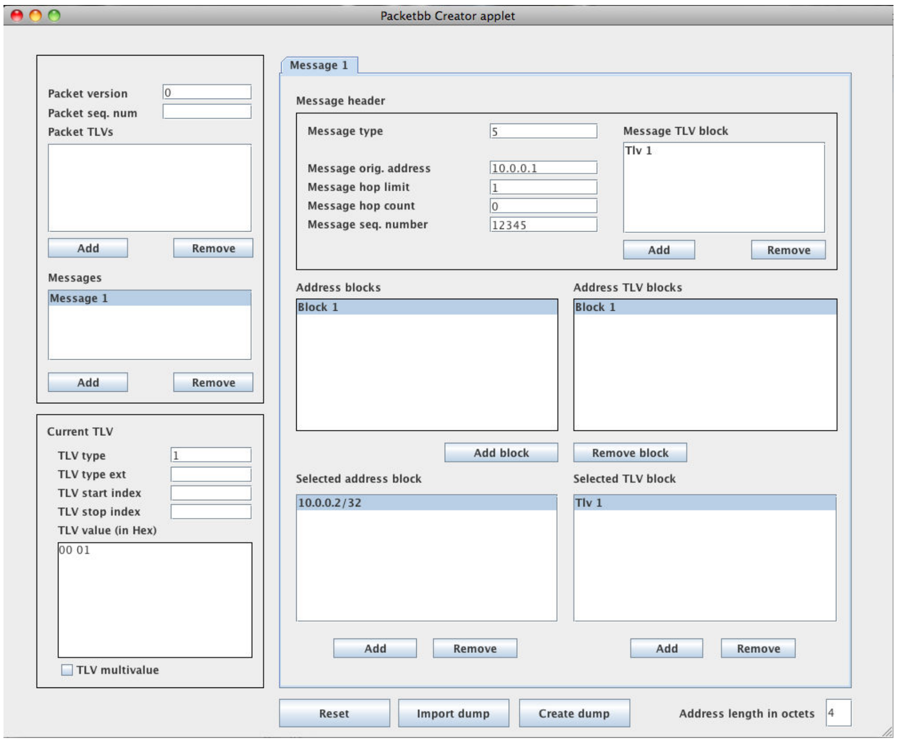


```
00 05
f3 00 1e 0a
00 00 01 01
00 30 39 00
04 05 10 01
05 01 00 0a
00 00 02 00
04 06 10 01
0a
```

```
,------------------
|  PACKET
|------------------
| * Packet version:    0
| * Packet flags:  0
|     ,-------------------
|     |  MESSAGE
|     |-------------------
|     | * Message type:       5
|     | * Message addr. length:       4
|     | * Message flags:  240
|     | * Originator address: 10.0.0.1/32
|     | * Hop limit:          1
|     | * Hop count:          0
|     | * Message seq number: 12345
|     | * Message TLV Block
|     |     - TLV
|     |         Flags = 16
|     |         Type = 5
|     |         Value = 05
|     | - Address block (1 addresses)
|     |     - 10.0.0.2/32
|     |     - Flags = 0
|     | - ADDRESS TLV block (1 TLVs)
|     |     - TLV
|     |         Flags = 16
|     |         Type = 6
|     |         Value = 0a
|     `-------------------|
`------------------

         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1       |0 0 0 0 0 0 0 0|  Msg type=5   |1 1 1 1 0 0 1 1|          Lengt|
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
2       |h=30           |                    Orig-Addr=10.0.0.1/32      |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
3       |               |  Hop-limit=1  |  Hop-count=0  |Message Sequenc|
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
4       |e Number=12345 |      Tlv Block Length=4       |  Tlv Type=5   |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
5       |0 0 0 1 0 0 0 0|   Length=1    |0 0 0 0 0 1 0 1|  Num-addr=1   |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
6       |0 0 0 0 0 0 0 0|0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
7       |0 0 0 0 0 0 1 0|      Tlv Block Length=4       |  Tlv Type=6   |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
8       |0 0 0 1 0 0 0 0|   Length=1    |0 0 0 0 1 0 1 0|
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7.8: Packetbb Parser Applet



Packet version: 0
Packet seq. num:
Packet TLVs:

Messages: Message 1

Current TLV
TLV type: 1
TLV type ext:
TLV start index:
TLV stop index:
TLV value (in Hex): 00 01
TLV multivalue ☐

Message header
Message type: 5
Message orig. address: 10.0.0.1
Message hop limit: 1
Message hop count: 0
Message seq. number: 12345
Message TLV block: Tlv 1

Address blocks: Block 1
Address TLV blocks: Block 1
Selected address block: 10.0.0.2/32
Selected TLV block: Tlv 1

Address length in octets: 4

Figure 7.9: Packetbb Generator

addresses from 10.0.0.2 to 10.0.0.5. These routers shall represent the 1-hop neighbors of 10.0.0.1. These four routers are not running on four different machines, but on a single machine with a single interface with four different IP addresses (using virtual interfaces in the case of IPv4). The emulator running on this machine creates HELLO messages with using these four IP addresses as source address in the IP header, and thus emulates four direct neighbors of 10.0.0.1. Note that the HELLO messages already include 10.0.0.1 in the list of symmetric neighbors, so that the link is symmetric starting from the first HELLO message. In addition, forwarded TC messages of "virtual" routers are created. These "virtual" routers represent routers that are at least two hops away from 10.0.0.1, but are not bound to an IP address on the emulator machine. The four 1-hop neighbors create TC messages and pretend that these originate from routers further away. For example, such a TC message might have a hop count of 4 (*i.e.* four hops away from 10.0.0.1), and any message originator address different from the four direct neighbors.

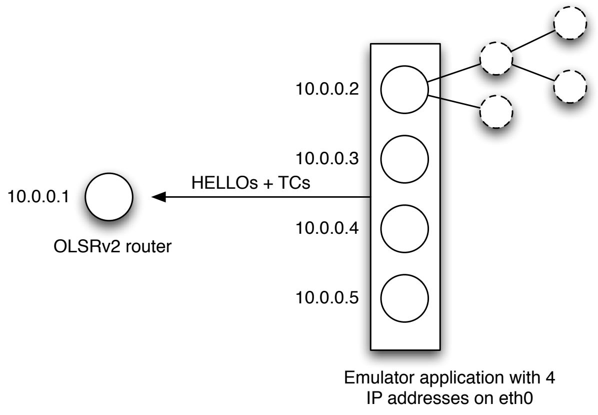


Figure 7.10: Emulator for creating large OLSRv2 topologies

The emulator thus represents arbitrary topologies to the router under test. An applet has been implemented that allows for creating topologies. In the Java applet, the user can "draw" routers by clicking on an area and creating edges between two routers (depicted in figure 7.11). Additionally, random graphs using the Eppstein Power Law [111] or Erdos-Renyi [112] can be created in the applet. As these graphs might be separated into several connected components, the applet connects these components randomly. Another feature is to import NS2 Tcl files representing a static scenario (*i.e.* not including any router movements). After having created the graph, a spanning tree with the router 10.0.0.1 being the root is calculated, and the emulator can send HELLOs and TCs that correspond to the topology of that graph. The emulator only allows to use static graphs, *i.e.* it allows to test incrementally adding edges in the routing tree, but not yet removing edges, in order to emulate dynamic scenarios (due to either mobility of routers or due to unstable wireless links).

The emulator creates both IPv4 and IPv6 topologies.

Topologies created with the emulator can easily contain several hundreds of thousands of routers.

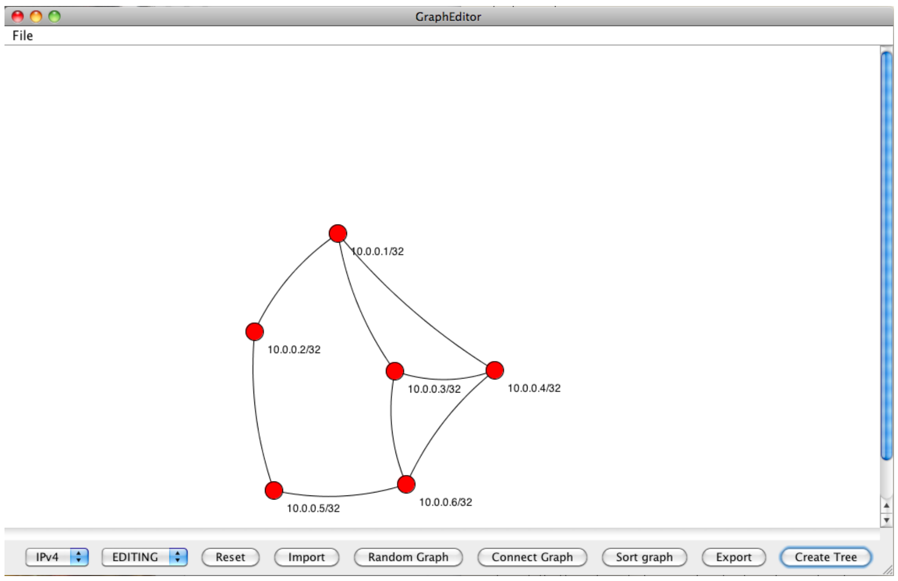


Figure 7.11: Topology creator for OLSRv2 topologies

7.7.4 NHDP Tests

Several tests have been developed to test the correct functionality of NHDP implementations, in part using the topology emulator described in section 7.7.3.

1. 30 HELLO messages are sent, each advertising 30 two-hop neighbors. This test allows for verifying whether the 900 neighbors and two hop neighbors are correctly inserted into the information bases of an NHDP implementation, and whether the implementation scales well enough to support that many entries.

2. 260 HELLO messages are sent, emulating 260 direct neighbors. This allows to test correct behavior of the HELLO message generation; if an implementation only puts all neighbors in a single address block, the maximum allowed number of addresses per address block of 255 is surpassed and the implementation might crash.

3. HELLOs are sent with random transmission errors, *i.e.* for each byte of a transmitted HELLO, the byte is changed to a random value with a probability of p. Such messages may be malformed, should be rejected by an implementation and should not lead to crashes.

4. HELLOs are syntactically correct, but the semantics are wrong, *e.g.* required TLVs are missing. An implementation should treat such messages according to the specification.

5. If triggered HELLOs are enabled, the correct HELLO intervals can be tested: the test application sends several HELLOs in a short interval, emulating several neighbors. The tested implementation may trigger HELLO messages. The test checks the correct adherence of the implementation to `HELLO_INTERVAL` and `HELLO_MIN_INTERVAL`.

6. Change of parameters, such as message intervals, during runtime should be possible, and can be verified by manual inspection.
7. Test of address and interface addition and removal during runtime should be possible, and can be verified by manual inspection.
8. Sequence numbers should wrap around correctly.
9. The test application sends a HELLO advertising several interfaces with several IPv6 addresses; the NHDP implementation should correctly treat the message and store the addresses correctly in the link sets.

7.7.5 OLSRv2 Tests

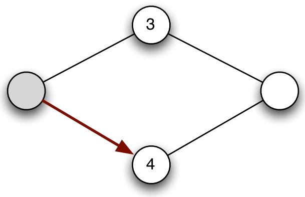
Several tests have been developed to test the correct functionality of OLSRv2 implementations, in part using the topology emulator described in section 7.7.3.

1. In order to test the correct behavior of the MPR selection, a topology depicted in figure 7.12(a) is emulated. The tested implementation should select the neighbor as MPR with the higher WILLINGNESS (in the example, the neighbor with WILLINGNESS of 4). The test application then transmits HELLOs according to the emulated topology depicted in figure 7.12(b), which should lead to a change of the selected MPR. The correct behavior of the OLSRv2 implementation can be verified by inspecting the emitted HELLO messages which advertise the selected MPR.
2. INCOMPLETE TC message test: the test application first sends a complete TC message advertising some neighbors. It then sends an incremental (“incomplete”) TC message. The OLSRv2 implementation should correctly add the newly advertised neighbors from the second TC message.
3. ANSN sequence number wrap around must work correctly (verified by manual inspection).
4. Routing table calculation test: several topologies of different sizes are emulated, using the topology emulator presented in section 7.7.3. The OLSRv2 implementation should correctly add all emulated routers in the routing table and calculate correct paths (verified by manual inspection).

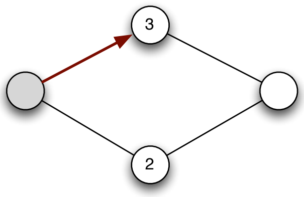
7.7.6 Packetbb Interoperability Test Results

The packetbb tests, presented in section 7.7.1, have been conducted with several implementations during the OLSR Interop Workshops in 2008 and 2009 [108]. The tested implementations were:

1. **JOLSRv2**: implementation presented in this chapter.
2. **OLSR.org packetbb**: The developers of the OLSR implementation available at www.olsr.org have implemented a packetbb implementation in C, available at [113]. An implementation of NHDP is announced to be available soon.



(a) The gray router selects the router with WILLINGNESS of 4 as MPR



(b) The gray router selects the router with WILLINGNESS of 3 as MPR

Figure 7.12: Willingness test: the numbers associated with the direct neighbors of the gray router represent the "WILLINGNESS", the bold arrows show the MPR selection of the gray router.

3. **NRL packetbb/NHDP**: Implementation of packetbb and NHDP from the Naval Research Laboratory, available at [114].

4. **packetTT**: packetbb implementation from "TNO Information and Communication Technology", code unavailable for public use.

5. **CRC packetbb/NHDP/OLSRv2 implementation**: described at [115].

Other implementations, such as nOLSRv2 [116] and an OLSRv2 implementation from Hitachi could not be tested because nOLSRv2 was not updated to the latest specifications and the code of Hitachi is not available for public use. Table 7.2 shows the test results from the initial packetbb interoperability test, as presented in section 7.7.1. If an implementation was able to parse correctly all tests, it is marked with a *p*. If it can also generate the same messages, it is marked with a *g*. Some implementations only have a *p* and not *g*, which does not necessarily mean that they cannot generate correct messages. But since implementers had to create the exact same packets from the test, not all were able to implement the tests in the time during the OLSR Interop workshop. As for NRL packetbb, the packetbb tests have not been completed. However, a test of NRL NHDP was performed with an emulated topology using the topology emulator presented in section 7.7.3. Moreover, a "multiple interface" test was conducted using the network emulator "CORE", which is described in section 7.7.7. The NRL packetbb code is also used to log HELLO and TC messages produced by JOLSRv2 in the NS2 simulator. Thus, it is reasonable to believe that NRL packetbb is compliant with the specification.

The implementations have not yet been tested using the NHDP and OLSRv2 tests, presented in sections 7.7.4 and 7.7.5.

Table 7.2: packetbb interoperability test results

JOLSRv2	FunkFeuer packetbb	NRL packetbb	packetTT	CRC packetbb
p/g	p/g	?/?	p/g	p

7.7.7 Network Emulation using CORE

The Common Open Research Emulator (CORE) [117] is a tool developed by the Naval Research Laboratory which allows to emulate entire networks on one or more machines. CORE consists of a GUI for easily drawing topologies that drives virtual machines, and various utilities. CORE uses virtualized network stacks in a patched FreeBSD kernel, or Linux virtual machines [117]. An example topology is depicted in figure 7.13. Machines can be added and equipped with one or

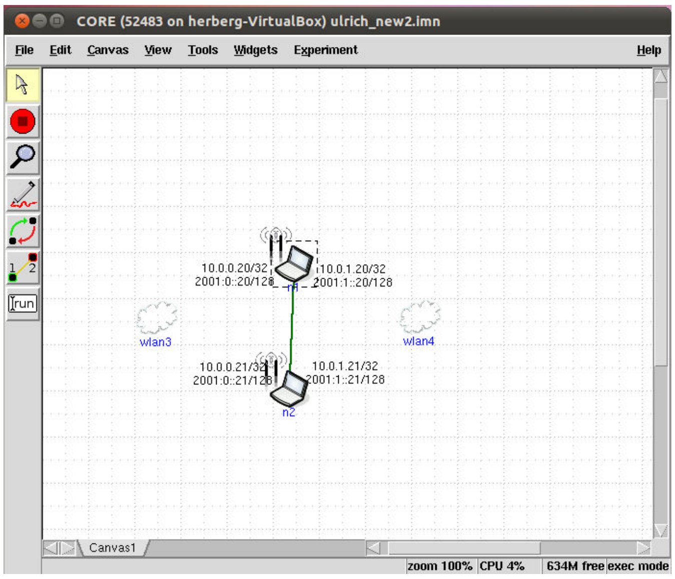


Figure 7.13: Interoperability tests with the network emulator CORE

more interfaces, each with one or more IP addresses. Each machine is virtualized and a terminal can be opened for each machine, where all interfaces and routes can be used as on a real Linux machine. The network stack can also be combined with a “real” network.

In an interoperability test, the NHDP part of JOLSRv2 and NRL-NHDP have been successfully tested on two machines, in the depicted topology in figure 7.13, with two interfaces on each machine. By “moving” the machines apart in the GUI, the connection is interrupted and NHDP will detect the link breakage (after the timeouts expire). Automated mobility patterns can also be defined in CORE. While CORE is a useful tool to test implementations, it does currently not replace network simulations, as the link characteristics are not modeled (*i.e.* either two machines are set to be in range, in which case no transmissions are lost, or the machines are out of range). However, it is planned to also emulate characteristics of, *e.g.*, 802.11 links.

7.7.8 Remote Tests

All interoperability tests presented in this section have been conducted on the same machine or machines on the same link. In order to perform “remote” tests via an Internet connection, the test application presented in sections 7.7.1 to 7.7.5 has been integrated into a Java servlet. The servlet runs on a server with a Tomcat [118] web server and can thus be accessed through a web browser. The web site allows to select the different tests, and then executes the chosen test, transmitting the emulated topology. For emulating arbitrary topologies, the topologies that are created with the topology creator, presented in section 7.7.3, can be uploaded to the web server and used in the emulation test.

7.8 Conclusion

OLSRv2 has been implemented in Java, allowing for (1) easy prototyping, (2) easily adding extensions due to object oriented programming, (3) integration of tools as Java applet, and (4) platform independence. The implementation, denoted JOLSRv2, is modularized into several components according to the specifications of the packet and message format (packetbb), the neighborhood discovery (NHDP) and the routing protocol OLSRv2. Throughout this manuscript, JOLSRv2 has been extensively used for simulations and evaluations of OLSRv2 and extensions thereof.

Several tools for interoperability have been presented, along with the results of successful interoperability tests with other packetbb, NHDP and OLSRv2 implementations. These tests allow to confirm the correct behavior of an implementation, and interoperability between several different implementations.

The contributions of this chapter have been published in [46].

Chapter 8

Dynamic Shortest Path Algorithm in OLSRv2

As described in section 6.3.2, OLSRv2 scales better in terms of message overhead than a classic link state routing protocol due to the MPR mechanism. But since OLSRv2 is supposed to run on constrained devices, such as the routers in the FunkFeuer network (with only 200 MHz and 16 MByte of RAM), the in-router memory and CPU requirements for calculating shortest paths to all destinations in the routing domain may also limit the scalability of the protocol.

OLSRv2 uses a variation of the well-known static Dijkstra algorithm for calculating paths to all destinations in the network. This implies that for each incoming HELLO or TC message with a new topology information, the previously calculated paths are deleted and a new graph is calculated on the router. This is particularly expensive in terms of CPU time in MANETs, due to the ad-hoc nature of MANETs as the topology may be subject to frequent changes. Additionally, the destinations advertised in control messages are received incrementally on each router, and only a small number of addresses are advertised in each control traffic message.

This chapter analyzes the use of a dynamic (*i.e.* incremental) shortest path calculation algorithm (DSP), and presents an evaluation of OLSRv2 with such a DSP. The evaluation compares the CPU time for calculating shortest paths in a large emulated network between the standard static algorithm and a DSP. This chapter shows that large-scale MANETs benefit by using a DSP algorithm in OLSRv2, and that the flexible structure of the OLSRv2 specification allows to do so without breaking compatibility with the specification.

8.1 Chapter Outline

Section 8.2 gives an overview of related work to the topic. Section 8.3 describes how updates to calculation of paths to all destinations in the MANET is specified in OLSRv2, and that the proposed extension in this chapter keeps compatibility with the OLSRv2 specification. Section 8.4 presents the methodology applied to measure the resource requirements of the proposed extension. In section 8.5, the results of the emulation are presented and explained. Section 8.6 evaluates the number of incremental routing recalculations using the NS2 simulator. Section 8.7 concludes the chapter.

8.2 Related Work

Extensive research about increasing efficiency of calculating the shortest path problem exists, notably by using incremental tree calculations. [119] compares 26 different algorithms for solving the shortest path problem. That survey concludes that DSP algorithms perform much faster then repeating static ones (such as Dijkstra). In addition to a theoretical comparison of the complexities of each algorithm, the survey provides experimental results of the required CPU time by each algorithm in a simulator, but not embedded in a real routing protocol. [120] is a similar comparison of dynamic shortest path algorithms, based on experimentation.

[119] suggests that the algorithm that proves to be the fastest in the survey is [121] when using the same data structures as in Bellman-Ford. [121] – henceforth called NST+ by using the first letters of the three authors – is an incremental shortest path algorithm. That means that the algorithm incrementally adds and removes new edges to and from the tree, instead of recalculating the whole tree for every new or removed edge. The paper proposes an algorithmic framework that allows to characterize a variety of dynamic shortest path algorithms including dynamic version of the well-known Dijkstra, Bellman-Ford, and D'Esopo-Pape algorithms, and shows their efficiency in simulations. [121] proves that the complexity of the NST+ algorithm is $O(D_{Max}|\delta_d|^3)$ where D_{Max} is the maximum out-degree of a vertex of a given network, and δ_d denotes the set of affected vertices. NST+, as well as more recent dynamic shortest path algorithms (such as [122] and [123]) claim that using such an algorithm can be very efficient in link-state routing routing protocols like OSPF and IS-IS. However, their evaluation was not based on an OSPF or IS-IS implementation.

[124] describes an implementation of a DSP algorithm in OSPF based on the Quagga open source routing software. The authors show that their DSP algorithm performs better than the default routing algorithm that runs on a particular Cisco router.

However, to the best of the author's knowledge, no study exists for using DSP algorithms in MANET routing protocols such as OLSRv2 [39]. Due to the dynamic natures of these networks, the topology typically changes much more frequently than in wired networks running OSPF (described in chapter 1). In addition, MANET routers usually have less CPU power and memory than routers running OSPF. This makes MANET routing protocols an ideal environment for using DSP algorithms.

8.3 Routing Set Updates in OLSRv2

While the specification of OLSRv2 [39] guides the implementers closely by detailed descriptions, it keeps the principle of "no-one-size-fits-all". As in MANETs very different deployments can be possible (static or mobile routers, different sizes of the network, different hardware for the routers, etc.), OLSRv2 allows freedom in order to facilitate different deployments. Amongst other, the choice of the routing calculation algorithm is bound by [39] only by:

> “The Routing Set MUST contain the shortest paths for all destinations from all local OLSRv2 interfaces using the Network Topology Graph. This calculation MAY use any algorithm, including any means of choosing between paths of equal length.“

An example of such algorithm – a variation of Dijkstra’s algorithm – is presented in the appendix of [39].

This means that using a DSP algorithm does not break with the OLSRv2 specification, as long as it provides shortest paths to all destinations given by the topology.

8.4 Test Methodology

This section describes the methodology used for evaluating the performance of using a DSP algorithm in OLSRv2.

8.4.1 Topology Emulator

The topology emulator, presented in section 7.7.3, has been used to create large topologies of up to 10000 routers.

8.4.2 OLSRv2 Implementation

For the evaluation, JOLSRv2, as presented in chapter 7, has been used. JOLSRv2, per default, implements the shortest path algorithm (a variation of the standard Dijkstra algorithm) proposed in the appendix of [39]. In order to compare the standard algorithm with the DSP algorithm, the Java class representing the routing set has been specialized and the NST+ algorithm [121] has been implemented. NST+ was chosen as an instance of a dynamic shortest path algorithm; other such algorithms (as for example [123]) could be used for an evaluation as well. Due to their similar properties (*i.e.* dynamically adding or removing edges), it is supposed that the order of magnitude is similar.

8.4.3 Time Measurement

In order to compare the static shortest path algorithm with the DSP algorithm, the accumulated time for calculating the routes is measured in each respective algorithm. Whenever the method for updating the routing set is entered, the time is saved in a variable, and the difference of time when leaving the method is added to a cumulative time counter. This, naturally leads to some inaccuracies, since JOLSRv2 is a multi-threaded application, *i.e.* other time-consuming threads might run in the background. However, in order to reduce the degree of inaccuracies, the following settings have been used:

- The validity time of HELLO and TC messages is set such that Link Tuples and Router Topology Tuples do not expire during the experiment, and thus no further actions (*e.g.* tuple expiration or Routing Set recalculations) have to be performed by the router.
- JOLSRv2 allows to disable sending TC and HELLO messages on a router. During the experiment, the router was not sending any control messages – neither periodic nor triggered – but only listening to the HELLO and TC messages from the emulator.
- The emulator sends the control messages only once. Thus, once the messages are received and processed by the OLSRv2 router, no further messages have to be processed. Thus, no CPU time is needed for further message processing.
- No other time-consuming processes run on the machine.
- The measurement is averaged over 20 runs.

8.5 Evaluation

This section presents the results of the evaluation. Figure 8.1 depicts the average time consumption for calculating the routes to all destinations in the emulated network. Random graphs have been created using the Eppstein Power law, from 10 to 10000 routers. The JOLSRv2 routing protocol implementation was tested on an Intel Core2 with 2.13 GHz and 4 GB of RAM with no other time-consuming processes running in the background.

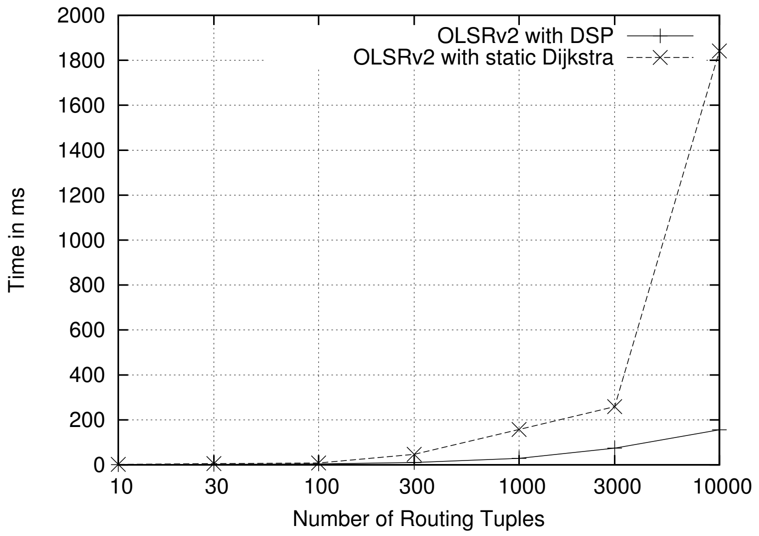


Figure 8.1: Results of the simulation: Accumulated time for calculating routes to all destinations

As can be seen in figure 8.1, for less than a few hundred routers, the difference of calculation time between the static Dijkstra and the DSP algorithm are negligible. Starting with about 100 routers, the static algorithm takes significantly longer than the DSP which stays at low values (in average 156 ms for 10000 routers in the given topology).

It can be assumed that in C++ implementations with intelligent memory management, the time consumption could be further reduced as compared to the Java implementation that was used in this comparison. Further improvements can also be achieved by using more recent DSP algorithms than NST+. For example, [123] claims to be up to three times faster than NST+.

8.6 Incremental Route Updates

Section 8.5 has shown that using a DSP algorithm for calculation of the shortest paths in OLSRv2 is beneficial for large networks. Since DSP algorithms incrementally add or remove edges in the shortest path tree instead of recalculating the whole tree for every Routing Set update, they are particularly suitable for networks with many changes of the topology, such as in mobile ad hoc networks. In static networks or in networks with few changes of the topology (such as typically in OSPF or IS-IS networks), DSP algorithms are less advantageous over static Dijkstra than in mobile networks.

This section analyzes the effect of mobility on the number of Routing Set updates in OLSRv2 and the number of routes that are changed in each Routing Set update.

8.6.1 Evaluation Settings

In order to evaluate the nature of Routing Set updates in MANETs using OLSRv2, a simulation with JOLSRv2 in the NS2 simulator has been performed, using the settings as presented in table 8.1.

Table 8.1: Simulation settings

Parameter	Value
NS2 version	2.34
Mobility scenarios	Random way point
Grid size	1000m by 1000m
Number of routers	40
Communication range	250m
Pause time	0 secs
Router velocity	0 to 30 m/s (constant)
Radio propagation model	Two-ray ground
Simulation time	100 seconds
Iterations	20 times
HELLO interval	2 secs
TC interval	5 seconds
Interface type	802.11b
Frequency	2.4 GHz

In the evaluation, only updates to the Routing Set that change (*i.e.* add, delete or modify) at least one Routing Tuple are counted. In addition to the number of Routing Set updates during the simulation, the number of changed routes in each Routing Set update is also considered.

8.6.2 Results

This section presents the results of the network simulation. In the following, only results from standard OLSRv2 are presented, since the same results are expected when using an incremental Dijkstra: as the scheduled interval of control messages and the number of advertised neighbors per control message does not change when using a DSP, the frequency of route recalculations, as well as the number of updated routes per control message will not change.

Figure 8.2 depicts the average number of Routing Set recalculations in standard OLSRv2 (that change at least one route) per router per second. As can be seen, for static networks few updates are performed. This is due to the fact that the Routing Set is updated only at the beginning of the simulation. Once the network has converged, no more updates are performed. The more mobile the network is, the more Routing Set updates have to be performed due to the changing topology. Figure 8.2 shows that for mobile networks, several updates of the Routing Set are performed per second, increasing with higher mobility.

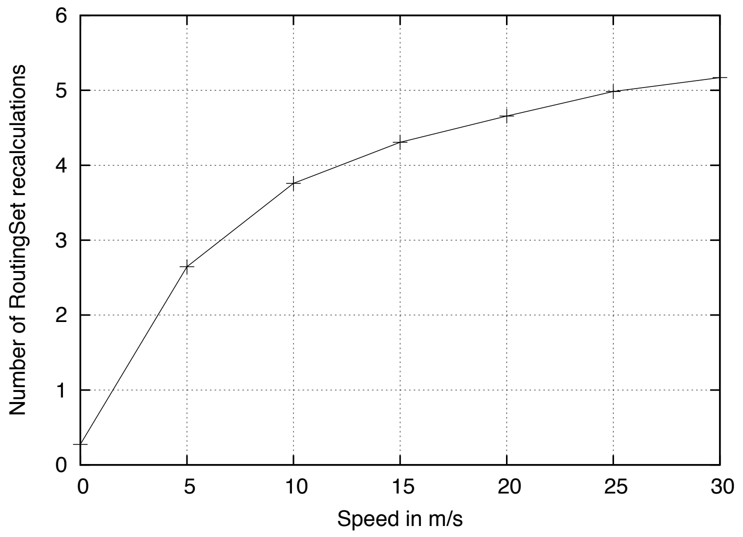


Figure 8.2: Number of Routing Set recalculations in OLSRv2, averaged per router per second

Figure 8.3 depicts the average number of updated routes in each of the Routing Set updates. Slightly growing with mobility, only few routes are modified each time the Routing Set is updated. This is beneficial when using an incremental DSP algorithm which keeps the previously unchanged routes in memory, and only adds or removes edges.

Figure 8.4 presents the number of control messages that are sent by each router in average over the simulation time of 100 seconds, both triggered HELLOs and TCs and total HELLOs and TCs (*i.e.* triggered plus periodic). The amount of control traffic is increasing with the mobility of the routers due to the triggered messages, which are emitted each time a router senses a change in its symmetric neighborhood (for triggered HELLOs) and changes in its MPR selectors (for

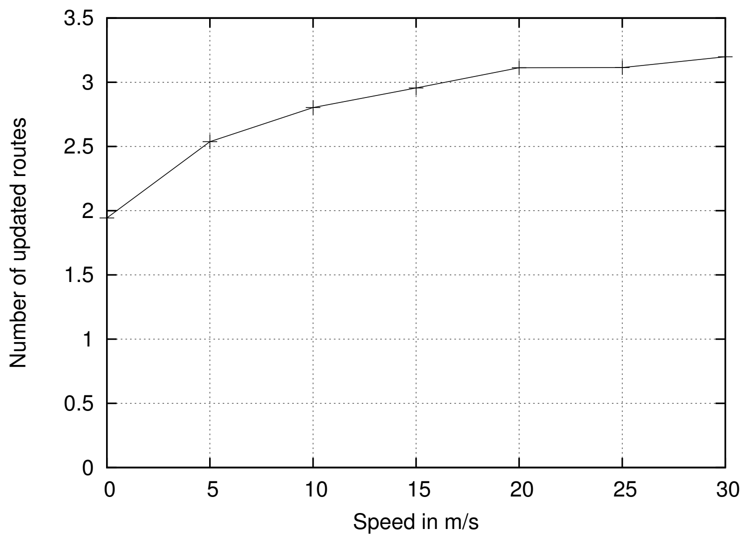


Figure 8.3: Average number of updated routes in OLSRv2 per Routing Set recalculation on a router

triggered TCs). Each incoming control message on a router can potentially lead to a Routing Set update if the topology has changed.

Figure 8.5 depicts the number of advertised neighbors in each control message. Each of such advertised neighbors has potentially to be considered when recalculating the Routing Set if the topology has changed. Due to the MPR mechanism of OLSRv2, TCs messages only advertise a subset of the emitter's neighbors in order to reduce redundant transmissions. As can be seen, the number of advertised neighbors in TCs is rather small, not significantly growing with mobility (since the total number of routers stays the same when increasing mobility). Due to the transmission of triggered incremental TC messages in OLSRv2, the number of sent control messages grows from a static to a mobile scenario.

In summary, the topology of mobile ad hoc networks frequently changes with increasing mobility of routers. While routers send many control messages, only a small number of destinations is advertised in each such message. Consequently, a router must often recalculate its Routing Set, but only few routes will change in each recalculation. This is beneficial for incremental shortest path calculations which only consider changed edges, instead of recalculating the whole tree. Note that in scenarios with extremely high mobility, it is possible that most edges have to be modified in the routing tree – in this case, using a DSP may not be more efficient (or even less efficient) than recalculating the tree from the scratch using a static Dijkstra [121]. The investigation of such high mobility scenarios is out of the scope of this chapter.

8.7 Conclusion

In this chapter, it has been experimentally shown that the calculation of the routing table in the MANET routing protocol OLSRv2 is able to scale up to tens of thousands of routers when

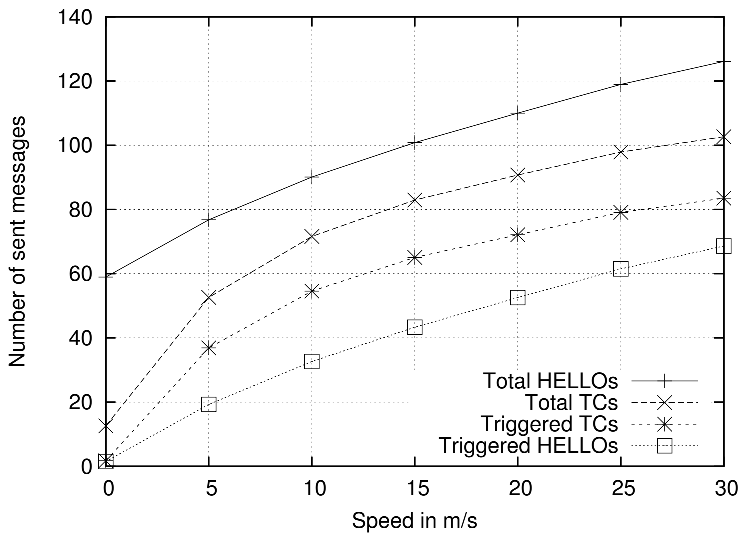


Figure 8.4: Total control traffic emitted by an OLSRv2 router during the simulation time of 100 seconds

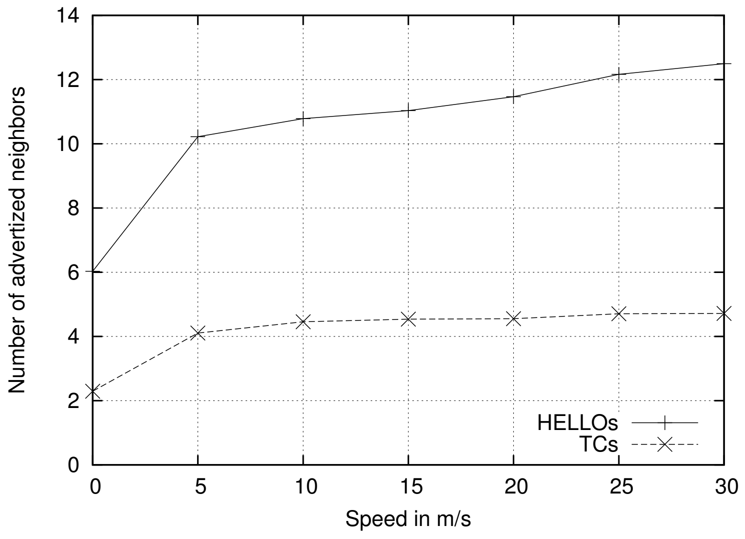


Figure 8.5: Number of advertised neighbors per control message

using a dynamic shortest path algorithm (DSP). A comparison has been performed between the static routing algorithm from the appendix of the OLSRv2 specification and a dynamic shortest path algorithm. The DSP takes significantly less CPU time for calculating paths, in particular for bigger networks with more than a few hundred routers. Due to the dynamic nature of MANETs, edges are incrementally added by means of control messages and change frequently. This is beneficial for the DSP, since it allows incremental updates from the previously calculated graph. Current community MANETs such as the FunkFeuer and FreiFunk networks reach this number of routers in their networks running OLSR. In addition, their routers are often low-power devices with limited CPU power and amount of memory. Thus, it is crucial to reduce the CPU time for path calculations in order to guarantee the scalability of these networks. This chapter not only has provided experimental justification of the efficiency of DSPs on a real OLSRv2 implementation, but also argued that due to the flexible nature of the OLSRv2 specification, full compatibility is kept when using a DSP algorithm instead of the provided algorithm in the specification.

The contributions of this chapter have been published in [47].

Chapter 9

OLSRv2 Network Management and Performance Monitoring

MANET routing protocols are commonly assumed to be entirely self-managing: routers, running such a protocol, perceive the topology of the MANET by means of control message exchange. Any change to the topology is reflected in the local routing tables of each router after a bounded convergence time, which allows forwarding of data traffic towards its intended destination. Usually, no human interaction is required, as all variable parameters required by the routing protocol are either negotiated in the control traffic exchange, or are only of local importance to each router (*i.e.* do not influence interoperability).

However, external management and monitoring of a MANET routing protocol may be desirable to optimize parameters of the routing protocol. Such an optimization may lead to a more stable perceived topology and to a lower control traffic overhead, and therefore to a higher delivery success ratio of data packets, a lower end-to-end delay, and less unnecessary bandwidth and energy usage. Such optimizations facilitate to scale the network to a large number of routers.

This chapter proposes a management framework to manage and control performance related objects on MANET routers running OLSRv2, and analyzes the performance of the "Simple Network Management Protocol" (SNMP) in OLSRv2-based MANETs when the network size and router mobility increases.

9.1 Chapter Outline

Section 9.2 describes general considerations for configuration of OLSRv2 routers. Section 9.3 gives a brief overview of SNMP. Section 9.4 outlines related efforts on management of MANETs as well as wireless sensor networks. Section 9.5 describes the motivation for the proposed SNMP-based management framework for OLSRv2-routed MANETs. The architecture of this framework is presented in section 9.6. Section 9.7 describes the construction and functioning of NHDP and OLSRv2 "Management Information Bases" (MIBs), while section 9.8 proposes the related

REPORT-MIB – a convenient tool for performance management. A performance evaluation of SNMP in OLSRv2-based MANETs is presented in section 9.9. This chapter is concluded in section 9.10.

9.2 OLSRv2 Router Configuration

The configuration of an OLSRv2 router consists of the set of prefixes "owned", and thus advertised, by the router, as well as interfaces of that router, participating in the OLSRv2 routing protocol. For each such interface, a set of parameters apply; other than the IP address(es) of each interface, these parameters consist of control message emission intervals, as well as the hysteresis values and link quality estimation, for setting the link status as described in section 6.3.1. It is important to note that agreement between OLSRv2 routers on the values for any of these is not required for interoperability. Link quality and hysteresis affect only which links a given router permits to become SYM or HEARD. Control message emission intervals and message content validity are encoded in outgoing control messages, by way of TLVs, such that a recipient router can correctly process these regardless of its own configuration.

9.3 A Brief SNMP Primer

Although it is recognized that the Simple Network Management Protocol (SNMP) [125] is not optimally designed for operation over MANETs, it is the preeminent management protocol for managing IP networks. As such, the management architecture for OLSRv2 routed MANETs is based here upon.

SNMP specifies a standardized way of exposing management data (system configuration, performance measurements, etc.) by way of defining a set of *objects* on the *managed devices*. These objects may then be read and, if appropriate, set in a standardized manner. This, by way of a *Network Management System* communicating with an *agent* on the managed device – in this case, an OLSRv2 router. SNMP does not mandate that a device must present a specific set of objects to read or set, but rather defines a standardized way in which a device may present such objects – a Management Information Base (MIB). A Structure of Management Information (SMI) defines modules of related management objects within such a Management Information Base.

Three versions of SNMP have been specified, developed and extensively deployed. Initially, SNMPv1 [125] specified a set of basic network management capabilities, including a relatively simple security model. SNMPv2 [126] was developed to extend SNMP capabilities and to improve the basic security model. However, it was not until the development of SNMPv3 [127] that an acceptable security model was developed [128, 129]. The Structure of Management Information version 2 (SMIv2) [130] is the current version of SMI. Using SMI, developers design and describe the management model for the system, protocol or device being managed. SMIv2 allows for the definition of fairly complex management models, yet allows for simplicity of chosen implementations through the definition of *Compliance statements* within the MIB.

It is useful to structure the MIBs according to a set of associated management functions, in order to provide a structured means of accessing the MIB. Further, it is useful to define the management objects, comprising the set of management functions, into basic management objects and more complex management objects. By doing so, the designer can specify through the Compliance statements a relatively simple version of the MIB for base management functions and a more complex and complete MIB for complete management of the managed device. An example where this structure may be useful is in the area of configuration management. It is common amongst most public IP carriers to perform configuration management through methods other than SNMP, *e.g.*, direct command line interfaces to remote devices. Therefore, following the outlined approach for structuring MIBs, one could define a compliant MIB which provided State, Performance and Fault Management functions while deferring Configuration Management to other methods.

9.4 Related Work

A number of papers analyze different approaches to monitor and to control MANETs as well as wireless sensor networks (WSN). Since SNMP [131] is the prevailing management system for networks, many solutions propose SNMP-derived protocols or SNMP extensions for MANETs and WSNs.

[132] proposes a distributed policy-based network management system for MANETs called DRAMA. Based on different policy levels, a hierarchy of management agents is established. In order to facilitate the hierarchy to systematically and autonomously self-form in MANETs, DRAMA enables routers to form “clusters” and then to link clusters into a tree-like hierarchy, with one cluster-head per cluster. The framework specifies a cluster maintenance system to assure keep-alives from routers within the cluster and to limit the size of the cluster within reasonable boundaries. DRAMA is compared to SNMP in a simulations study, which shows that DRAMA scales better than SNMP in MANETs (in terms of message overhead, timeliness of message delivery and message delivery ratio).

The authors of [133] claim that using proxy-SNMP has several limitations, and that therefore native SNMP should be supported. The paper describes an extended modification of the SNMP protocol for enabling 802.15.4 based networks (denoted 6LoWPAN) to provide efficient management capabilities, based on header and payload compression, as well as multicast SNMP messages and periodic message dissemination. Results have shown that using this extension, the overhead can be reduced by about 50% and that 6LoWPAN-SNMP can be successfully deployed in small sensor networks.

Another approach to allow management of constrained, mobile wireless routers, is [134]. The paper proposes a management tool for WSN based on SNMP: LiveNCM. LiveNCM introduces the concept of non-invasive context-awareness to diagnose the wireless sensor node state in order to reduce the network traffic and the local state on a router. Similar to [133], LiveNCM suggests compression of messages.

While SNMP typically allows to control and to manage a single router, some solutions propose to aggregate SNMP messages using multicast to several routers, such as [135]. This allows to control simple parameters of several routers with a single SNMP message, and also to monitor the state of the routers using aggregated messages. Similarly, [136] suggests a probabilistic scheme for managing only a subset of routers in the MANET, "such that we capture the most *interesting* nodes into the management", where *interesting* is defined with respect to relative good network presence and topology relationship.

Finally, several papers focus on managing MANETs based on particular routing protocols. [137] and [138] specify a MIB for the OLSR [79] routing protocol, allowing to set the protocol parameters and to monitor the state of the router (such as the neighbor and routing set tables).

9.5 Problem Statement: Managing OLSRv2 Networks

As indicated in section 9.2, OLSRv2 imposes minimal constraints on valid router configuration parameters, in order for OLSRv2 routers to interoperate.

Fundamentally, the only parameter upon which agreement is required is C – a constant, used to fix the scale and granularity of validity and interval time values, as included in protocol control messages. [36] proposes a value for this constant; the symbol C is chosen to indicate it to be a "constant of nature" inside an OLSRv2 network, to which all routers must adhere. As control messages carry validity time and interval time values, a recipient OLSRv2 router can behave appropriately, even if it uses vastly different values itself, as long as the recipient and sender use the same value for C.

Link admittance, by way of the hysteresis values and link quality estimation, requires no agreement; these are used for an individual router to determine a suitable threshold for "considering that a link *could* be a candidate for being advertised as usable".

Still, external monitoring and management may be desirable in an OLSRv2 network. A network may benefit from having its control message emission tuned according to the network dynamics: in a mostly static network, *i.e.* a network in which the topology remains stable over long durations, the control message emission frequency could be decreased in order to consume less bandwidth or less energy. Conversely, of course, in a highly dynamic network, the emission frequency could be increased from improved responsiveness. Concerning the hysteresis and link quality estimation, a management application might detect a region of an OLSRv2 network with a high link density – but also a high degree of "flapping": links coming "up" (SYM) only to disappear as LOST shortly thereafter. Detecting such behavior, on a global level and for multiple routers in the same region, could enable appropriately "tuning" the thresholds towards more stable links and, thus, a more stable routing structure in the network.

These are but two examples, and have as common that a more "global view" of the network, than that of a single OLSRv2 router, is required – *i.e.* entail that a *Network Management System* is able to inquire as to various performance values of the network, and to set various router parameters.

Thus, a first-order task is to identify suitable management data for an OLSRv2 routed MANET, and to describe these by way of MIBs for use by an SNMP Network Management

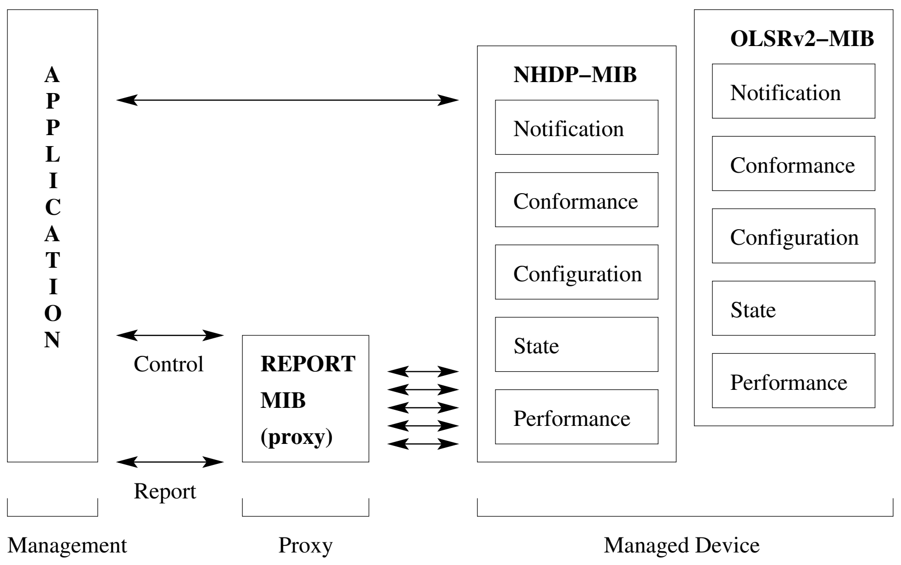


Figure 9.1: The OLSRv2 management model

System.

In the following sections, the proposed MIBs for managing OLSRv2 networks and monitoring performance of these networks are described in detail.

9.6 OLSRv2 Management Architecture

The proposed architecture of the OLSRv2 management system is depicted in figure 9.1. As is standard for SNMP management architectures, a Network Management System interacts with the various components of the device models directly over the network. However, frequent polling for object values in such a system involves a frequent and bandwidth-consuming message exchange. Further, due to highly variable network delays, it is not possible for a management application to determine the time associated with object values obtained via polling. In order to specifically address the issues associated with running SNMP for performance management over low bandwidth and high latency networks, typical of MANETs, the proposed performance management architecture is based upon a proxy capability, denoted REPORT-MIB [139]. This proxy is located locally on the managed devices and offers remote generation of performance reports established via the management application using Remote Monitoring (RMON) style control and reporting. The proxy then polls (locally) for the current values of the relevant objects necessary for the generation of the performance reporting.

9.7 NHDP and OLSRv2 MIBs

This section describes the design of the NHDP-MIB [49] and the OLSRv2-MIB [50]. As the protocols themselves are designed in a similar fashion, so are their associated MIBs. At the highest level, both the NHDP-MIB and the OLSRv2-MIB are organized into the following groups:

- Configuration Group – switches, tables, objects which are initialized to default settings or set through the management interface defined by the MIB.
- State Group – automatically generated object values which define the current operating state of the NHDP or OLSRv2 protocol process in the router.
- Performance Group – automatically generated object values which help an administrator or automated tool to assess the performance of the protocol process on the router and the overall performance within the routing domain.
- Notification Group – objects defining triggers and associated notification messages allowing for asynchronous tracking of pre-defined events on the managed device.
- Conformance Group – groupings of the above objects defining various levels compliance to the MIBs.

The Configuration Group for the NHDP-MIB and OLSRv2-MIB includes objects which control message intervals (*e.g.* for HELLOs), information validity times (*e.g.* hold times), link quality (*e.g.* thresholds to determine usefulness of the links), and message jitter. For the OLSRv2-MIB, additional configuration information include objects related to hop limits and routers' willingness measures to act as Multi-Point Relays (MPRs). Details on the actions these objects have on the respective protocols are found in [37] and [39].

Regarding the State Group, both protocols are defined in terms of the various databases developed by the protocols in order for their proper function. These (state) databases include the following set for the NHDP protocol:

- The Local Information Base (LIB), contains the network addresses of the interfaces (MANET and non-MANET) of the local router.
- The Interface Information Based (IIB), records information regarding links to a local MANET interface and symmetric 2-hop neighbors which can be reached through such links.
- The Neighbor Information Base (NIB), records information regarding current and recently lost 1-hop neighbors of the local router.

The OLSRv2 protocol extends the above databases, as well as defines the following additional databases:

- The Topology Information Base (TIB), records information used for the calculation of the Routing Set.

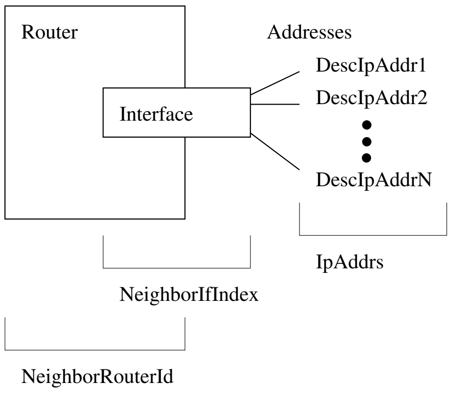


Figure 9.2: The linkage between the OLSRv2 and the NHDP MIBs

- The Received Message Information Base (RMIB), records information regarding messages, that have been previously received, processed, or forwarded by the local router.

The state tables in the OLSRv2 and NHDP MIBs are linked through two constructs (or TEXTUAL CONVENTIONS) developed within the MIBs as illustrated in figure 9.2. Within the NHDP and OLSRv2 protocol definitions, the various Information Bases provide information on discovered address sets, which are associated with discovered interfaces, which belong to discovered (or local) routers. These are used as indexes into the various State Tables; specifically as *IpAddr*, *DiscNeighborIfIndex* and *DiscNeighborRouterId*. And these objects are correlated through the *nhdpDiscIfSetTable* in the NHDP-MIB. Further, as the related State Tables rely on the same indexing, it is relatively straightforward for a network management application to cross-reference data from the two MIBs.

Finally, the MIBs define two levels of Conformance; a Basic Compliance which includes only Configuration Group objects and a Full Compliance which includes Configuration, State, Performance and Notification Group objects.

9.8 Performance Management

Apart from objects for monitoring and controlling parameters and data sets in NHDP and OLSRv2 (specified in section 9.7), a number of objects are proposed which permit to analyze the performance of NHDP and OLSRv2. This section describes the different types of objects and their intent for NHDP and OLSRv2.

9.8.1 Object Types

Some of the objects (denoted “base objects”) explicitly appear in the NHDP-MIB and OLSRv2-MIB while others are obtainable through a combination of base objects from the MIB and reports available through the REPORT-MIB.

The full list of base objects in the NHDP-MIB and the OLSRv2-MIB is comprised of different counters (*e.g.* for counting the total number of transmitted HELLO messages, number of changes to the neighbor set and to the 2-hop set, up-times of neighbor routers, etc.).

In order to infer performance problems in an OLSRv2 network, it may not be sufficient to access objects describing the total number of events, but objects describing the development of events over time. These objects, denoted "derived objects", are not specified in the NHDP-MIB and OLSRv2-MIB, but can be acquired using the REPORT-MIB. The REPORT-MIB allows to create reports "offline", possibly on another, more powerful device than the router running NHDP and OLSRv2. Notably, histories (based on timestamps) can be created over all of the performance related base objects.

For example, it is possible to create a histogram of intervals between transmitted HELLO messages, separated by periodic and triggered HELLOs. The histogram would display the distribution of intervals between two consecutive HELLOs of the same type (triggered or periodical) using a given bin size. Figure 9.3 depicts such a histogram.

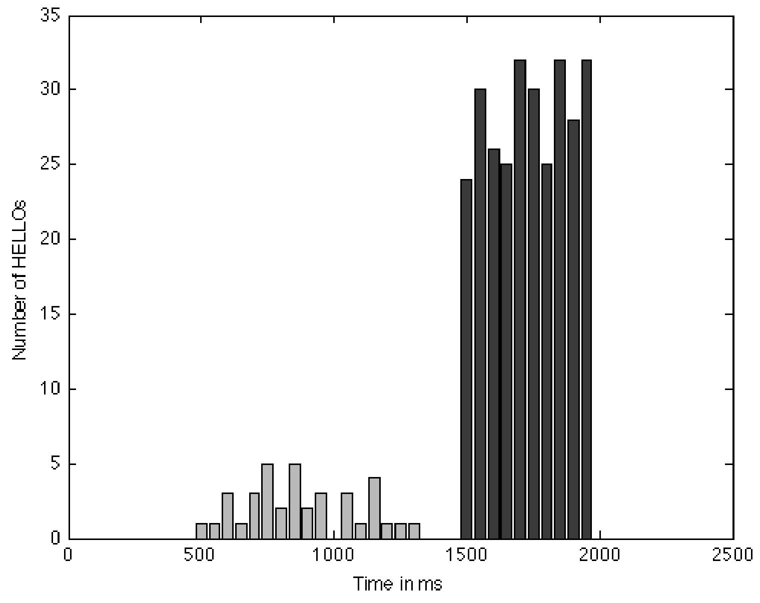


Figure 9.3: Example histogram: number of periodic (dark gray, between 1500ms and 2000ms) and triggered (light gray, between 500ms and 1500ms) HELLOs in 500s time, with a HELLO interval of 2000ms and a maximum jitter value of 500ms

Moreover, the NHDP and OLSRv2 MIBs in combination with the REPORT-MIB allow to display the changes of the frequency by displaying the changes of histograms over time. The total duration of recorded events is split into a given number of equal bins. Then, a histogram is created for each bin and the "distances" are calculated between each two adjacent histograms in time (*e.g.*, using the Bhattacharyya distance [140]). Note that while visualizing a change in the frequency of events may help the network administrator to understand changing properties of the network, it is out of scope of the MIB, and of this chapter, to automatically determine whether such a change indicates a performance problem or is part of the natural change of topology of the network.

9.8.2 Derived Objects in NHDP and OLSRv2

As described in section 9.8.1, changes of the frequency of certain events may indicate performance issues in the MANET. Notably unstable neighbors or 2-hop neighbors and frequent changes of sets may have a negative influence on the performance of NHDP and OLSRv2, wherefore a number of derived objects have been specified in the MIBs that allow management applications to acquire information related to the stability of NHDP and OLSRv2. The following list describes several derived objects from the MIBs that are relevant for NHDP and OLSRv2 networks:

9.8.2.1 Frequency Changes of Message Scheduling

A change in the message scheduling frequency can appear if, for example, suddenly many triggered HELLO or TC messages are sent, whereas only very few such triggered messages were sent in the past. This can indicate a sudden change in the topology experienced by a router.

9.8.2.2 Frequency Changes of Neighbor Set Modifications

This derived object allows to visualize the changes of frequency of neighbor set modifications. A neighbor set modification is defined as a new neighbor that is added, a neighbor that is removed, or a neighbor that changes its symmetry status. If, for example, there are five changes of the neighbor set per minute in average, and then this frequency is increased to 100 changes per minute, this can indicate a performance problem.

9.8.2.3 Frequency of Changes of the Online Status of a Given Neighbor

If a neighbor (identified by its IP address) changes its "online" status very frequently (*i.e.* a neighbor tuple for that neighbor is alternatively added and removed again in a very short time), this may indicate a performance problem.

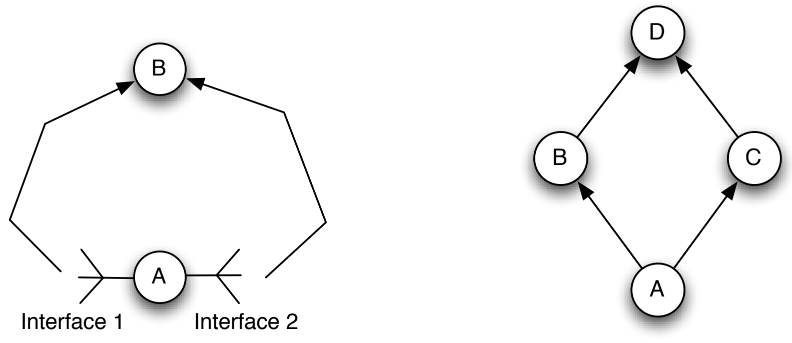
9.8.2.4 Frequency of Changes of the Online Status of a Given 2-hop Neighbor

Similar to the frequency of changes of the online status of a neighbor, a derived object in the MIB allows to track the frequency of change of the online status of 2-hop neighbors.

9.8.2.5 Frequency of Changes of the Link over which a Neighbor is Reachable

If a neighbor changes the interface over which it is reachable very frequently, that can cause performance issues: (i) more in-router resources for updating the internal data structures, and (ii) additional control traffic messaging may be required (*e.g.* when sending triggered HELLO messages).

The example in figure 9.4(a) depicts such a "flapping" of a neighbor between several links. Router A has two interfaces over which it can communicate with router B. If the corresponding link tuple for the neighbor frequently switches between interface 1 and 2, the above-mentioned performance issues may arise.



(a) Router flapping between several interfaces

(b) 2-hop router flapping between several neighbors

Figure 9.4: Flapping

Such flapping of a neighbor may, for example, stem from inappropriate hysteresis values of the link quality selection of NHDP, or from a random choice of equal length paths in the Dijkstra algorithm when recalculating routes. Analyzing the frequency of neighbor flaps facilitates to modify the values to stabilize the link formation and removal on the OLSRv2 interfaces.

9.8.2.6 Frequency of Changes of the Neighbor over which a 2-hop Neighbor is Reachable

Similar to the link-flapping of a neighbor as described above, a two-hop neighbor can flap between several one-hop neighbors, as depicted in the example in figure 9.4(b).

In addition to the in-router resource requirements for updating the internal data structures, this flapping between neighbors can – in the worst case – induce (i) routing set recalculation on all routers in the MANET, (ii) MPR set recalculations in the 2-hop neighborhood, and (iii) transmission of triggered HELLO and TC messages. The reason is that each time the 2-hop neighbor flaps between neighbors, a new MPR selection may be necessary. In the example, router A might initially have selected B as MPR in order to reach D. If A is forced to switch the MPR to C because B is not available any more, TC messages with the updated topology information will be disseminated throughout the network, forcing every router to recalculate its routing set.

9.8.2.7 Frequency of routing set recalculations and MPR set recalculations

The MIB provides two derived objects for observing routing-set and MPR-set recalculations over time. Both operations are costly in terms of in-router resources (such as memory and CPU time), and too frequent recalculations may reduce the life-time of the MANET when using battery-powered routers. The MIB objects allow an administrator to “tune” parameters of OLSRv2 in order to reduce the number of unnecessary recalculations.

9.9 Performance Study of SNMP in OLSRv2 MANETs

Surveys of performance aspects of SNMP exist [141]), yet – to the best of the author's knowledge – none which consider performance in MANETs. The reasons for the lack of research in this area may be twofold: (i) SNMP may be considered too "heavy-weight" for MANETs, and (ii) implementing a fully compliant SNMP for network simulators may take a lot of time. Concerning (i), it has to be noted that as no "light-weight" management protocol has been standardized by the IETF yet, SNMP remains the prevailing management protocol[1]. As to (ii), despite the 'S' in SNMP standing for "simple", it is composed by a many RFCs, and therefore it is a time-consuming task to reimplement SNMP for network simulators.

This section presents results of a simulation study of SNMP in an OLSRv2 MANET with the NS2 simulator. Typical performance metrics – such as delivery ratio, delay, overhead and collision ratio – are evaluated.

9.9.1 Simulation Settings

Simulations have been conducted with JOLSRv2 as routing protocol. SNMP4J [143], a Java implementation of SNMP, has been hooked into NS2 using AgentJ, as explained in chapter 3. According to [143],

> "*SNMP4J is an enterprise class free open source and state-of-the-art SNMP implementation for Java 2 SE 1.4 or later. SNMP4J supports command generation (managers) as well as command responding (agents). Its clean object oriented design is inspired by SNMP++, which is a well-known SNMPv1/v2c/v3 API for C++ [...].*"

Simulations have been performed using relatively standard scenario parameters, as depicted in table 9.1. Each measured value has been averaged over 10 simulation runs.

Table 9.1: NS2 parameters

Parameter	Value
NS2 version	2.34
Mobility scenario	Random walk
Area	1000m x 1000m
Number of routers	10 - 50
Communication range	250m
Radio propagation model	Two-ray ground
Simulation time	270 secs
Interface type	802.11b
Radio frequency	2.4 GHz
OLSRv2 parameters	Proposed default values of [39]

In all scenarios, one router (with ID of 0) is positioned at exactly the center of the simulated area (*i.e.* at coordinates (500, 500)) and does not move. It runs an SNMP manager, whereas all

[1]The IETF has standardized NETCONF [142] in 2006, but not with the focus on constrained devices such as MANET routers

other routers run an SNMP agent, providing the NHDP-MIB as described in section 9.7. During the simulated time of 270 seconds, the SNMP manager sends requests ("get-next-request") for the NHDP parameter `N_HOLD_TIME` to all other routers, one after the other, starting after 10 seconds (in order to allow OLSRv2 to converge before). UDP is used as transport protocol, and a 500 ms timeout is set (*i.e.* the manager aborts the request if no response has been received within 500 ms, and it proceeds sending a request to the next router). 25 seconds after the first request is sent, the manager restarts sending requests to all other routers, which allows to send requests to all routers (500 ms times 50 routers in the worst case of a 50 router network and all requests failed). The process is repeated every 25 seconds until the end of the simulation. Figure 9.5 depicts a sample simulated network with 20 routers.

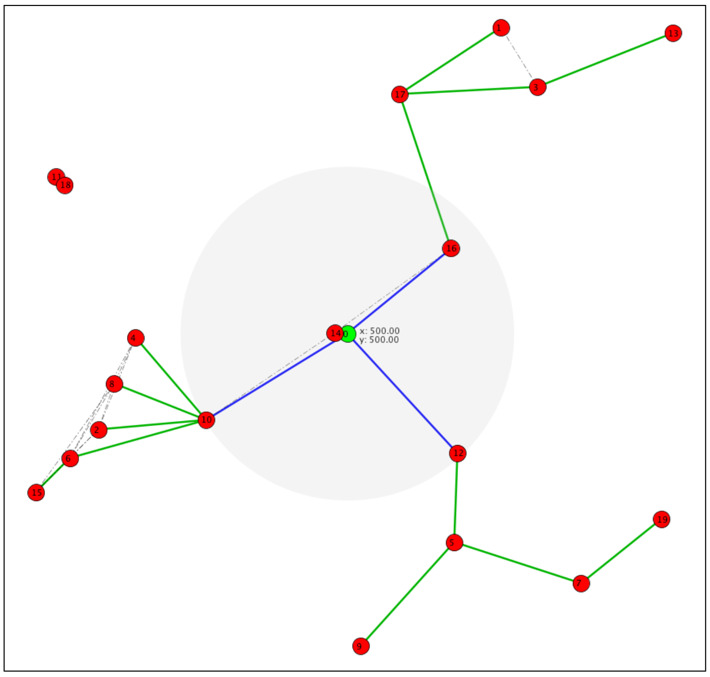


Figure 9.5: Sample MANET: Router '0' in the middle of the area is the SNMP manager with routes calculated by OLSRv2 to all other reachable routers, which run an SNMP agent

Different variations of SNMP are tested in the simulations, notably SNMPv2c, SNMPv3 without authentication or privacy (simply denoted "SNMPv3"), SNMPv3 with SHA authentication only (denoted "SNMPv3 (SHA)"), and SNMPv3 with authentication and privacy (denoted "SNMPv3 (SHADES)" and "SNMPv3 (SHAAES128)"). SHADES is specified in [128], and SHAAES128 in [144]. Note that some implementations, such as SNMP4J and Cisco SNMP implementations, provide other cipher algorithms like SHAAES192, SHAAES256 and SHA3DES. However, these have only been proposed as individual drafts (expired) in the IETF, and have never been standardized. For that reason, these mechanisms have not been considered in the simulation. MD5 as authentication mechanism has also not been considered, as it is considered deprecated [145].

9.9.2 Simulation Results

This section presents the results of the simulation study.

Figure 9.6 illustrates the accumulated overhead of SNMP in the given scenario compared to the control traffic of OLSRv2 during the simulation, counting each retransmission of forwarded messages in OLSRv2. The control traffic overhead increases with the number of routers in the network. It can be observed that the additional traffic incurred by SNMP is small compared to the OLSRv2 control traffic in the specific scenario.

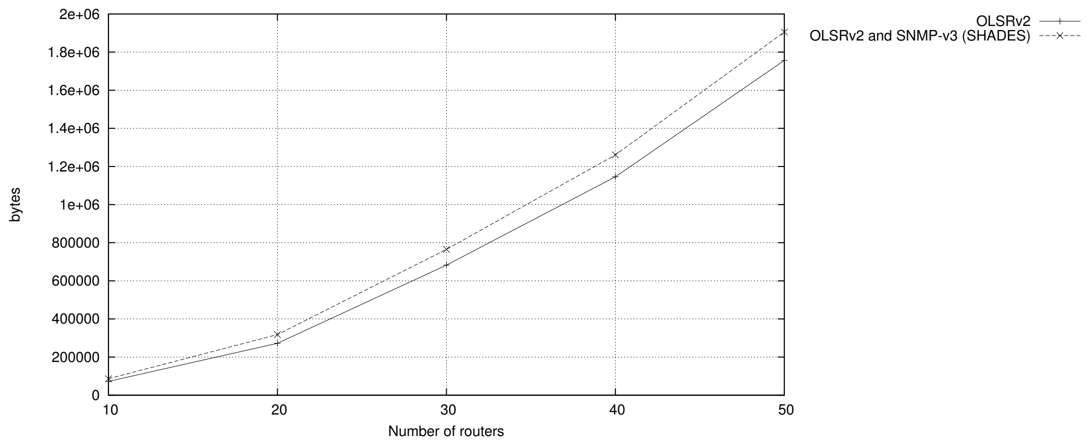


Figure 9.6: Comparison of accumulated traffic with OLSRv2 only, and OLSRv2 together with SNMP traffic throughout the simulation

Figure 9.7 presents the accumulated traffic of SNMP messages with different SNMP versions and security mechanisms. The traffic grows linearly with the number of routers in the network, *i.e.* with the number of SNMP agents to which the manager sends SNMP requests. Not surprisingly, SNMPv2 has a far lower overhead than SNMPv3. SNMPv3 with authentication only (SHA) has a higher overhead than SNMPv3 without authentication, but less than both tested encrypted SNMPv3 variants (which have an almost equal overhead).

The difference in the accumulated message overhead has several reasons: First, the SNMP messages contain different amount of security related parameters. Table 9.2 shows the message sizes of the get-next-request message that is sent from the manager to the agents (as measured with Wireshark [146] between two real machines, but the same SNMP implementation, SNMP4J).

Table 9.2: SNMP message sizes

Variant	Frame size	SNMP message size	PDU size
SNMPv3	140	103	48
SNMPv3-SHA	146	109	48
SNMPv3-SHADES	159	122	48
SNMPv3-SHAAES128	163	125	48

It can be observed that SHADES and SHAAES128 have very similar SNMP message sizes, which confirms the almost equal plots in figure 9.7. Another observation is that the payload (the

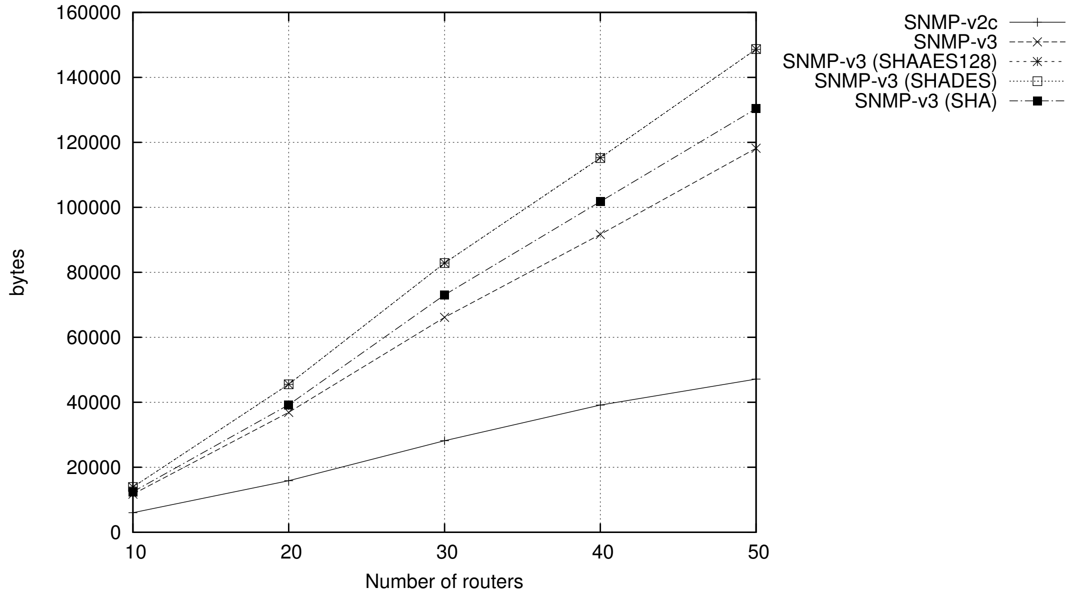


Figure 9.7: Accumulated SNMP traffic overhead during the 270 seconds simulation

PDU) is of equal size. This is due to the used mode of operation of the block cipher (for more details refer to [147]). SHADES applies a CBC (Cipher-block Chaining) operation mode, which splits the plaintext in multiples of 8 bytes with possible padding. Since the payload happens to be a multiple of 8 bytes, the cypher text has the same length as the plaintext. SHAAES128 uses a CFB (Cipher Feedback) operation mode, which always outputs the same length as the input plaintext.

Another reason for the different total SNMP traffic is the number of transmitted messages, which is depicted in figure 9.8. The figure compares SNMPv2c with the SHAAES128 variant of SNMPv3 (for the other encrypted variants, the output is similar). The reason for the higher number of messages is that between each pair of routers exchanging SNMP messages, an additional initial message exchange has to be performed, in order to provide a replay mechanism. Figure 9.9 shows an initial message exchange in SNMPv2 and SNMPv3 with privacy.

While in SNMPv2, the get-next-request is directly sent and answered, SNMPv3 exchanges the Authorative EngineID and a counter how often the agent has been rebooted, in order to provide replay protection. In the presented simulations, this initial exchange of parameters is only performed at the first request from the manager to an agent, not in any subsequent one, which explains why the plot in figure 9.8 for SNMPv3 is only slightly higher.

Figure 9.10 depicts the MAC frame collision rate. Due to the increased number of control traffic exchange and unicast traffic, the collision rate increases with the number of routers in the network. There is no significant difference between the different SNMP variants, since the main traffic in the scenario comes from the control traffic exchange of OLSRv2. It has to be noted that this is no general observation: in the simulated scenarios, only a single SNMP message exchange is performed at a time, and no other unicast traffic is added, which leads to low bandwidth

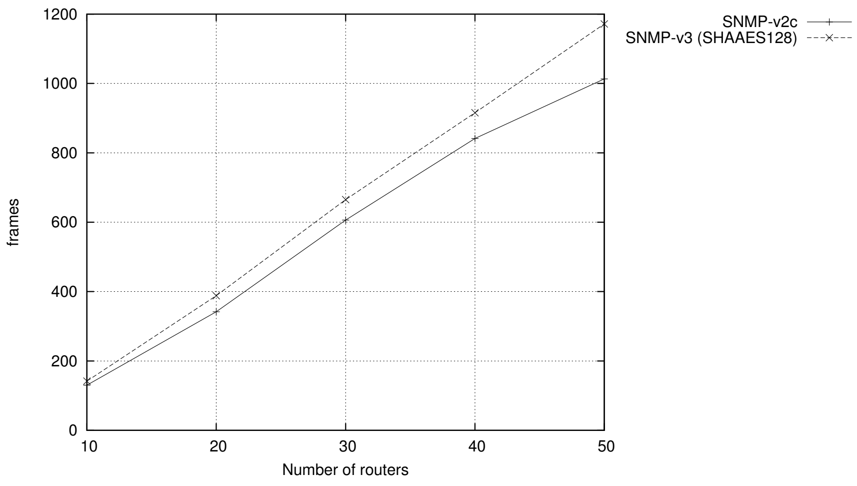


Figure 9.8: Number of transmitted SNMP messages

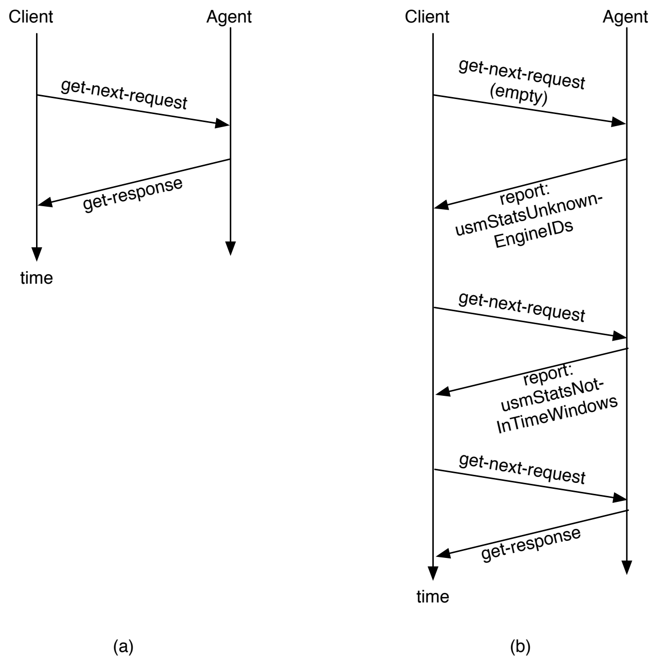


Figure 9.9: SNMP message exchange: (a) in SNMPv2 (b) in SNMPv3 with privacy

consumption of the unicast traffic.

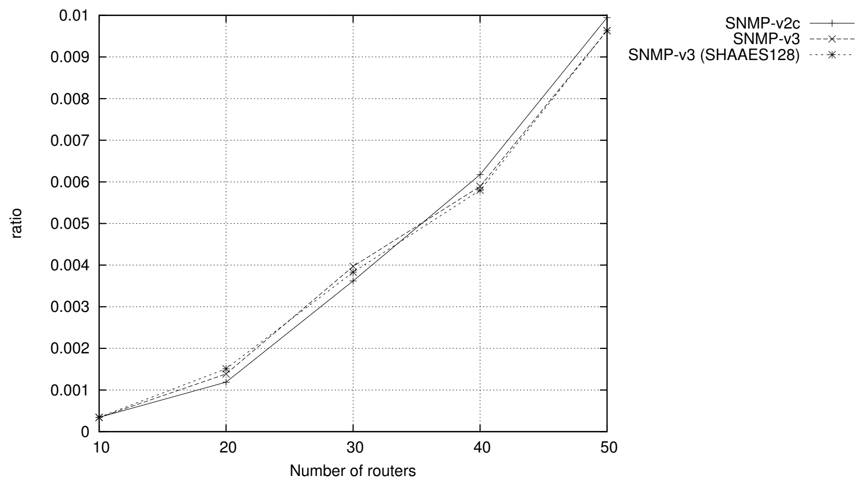


Figure 9.10: MAC collision ratio

Figure 9.11 illustrates the collision ratio with increasing speed of the routers. As the control traffic accounts for the majority of the traffic in the simulation, and the control traffic does not considerably increase with higher speed, the collision ratio remains at a stable level of about 12%.

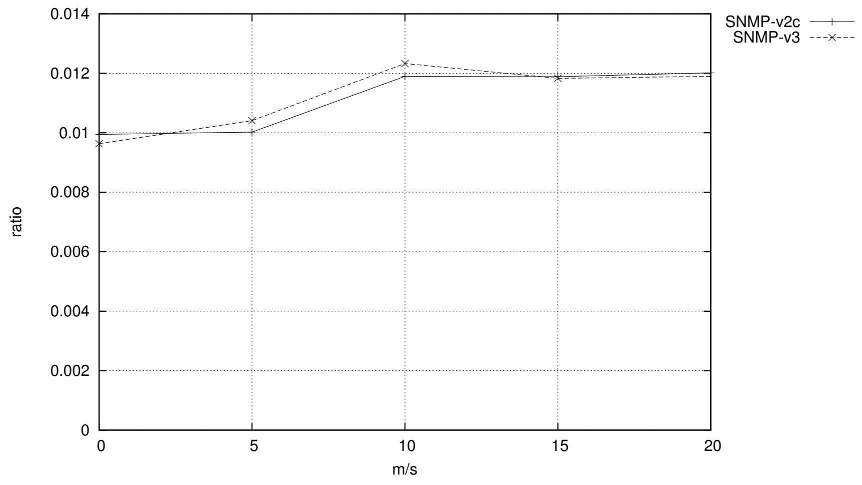


Figure 9.11: MAC collision ratio with variable router speed with 50 routers

Figure 9.12 shows the average time duration between a transmission of the get-next-request and the reception of a response. With increasing number of routers in the network, the delay increases in all SNMP variants. SNMPv3 and SNMPv3 with privacy has a higher delay due to the initial message exchange for means of replay protection as depicted in figure 9.9.

Figure 9.13 shows the delivery ratio of SNMP messages. Since the collision ratio is low (as depicted in figure 9.10), the delivery ratio is relatively high, increasing with a higher density of

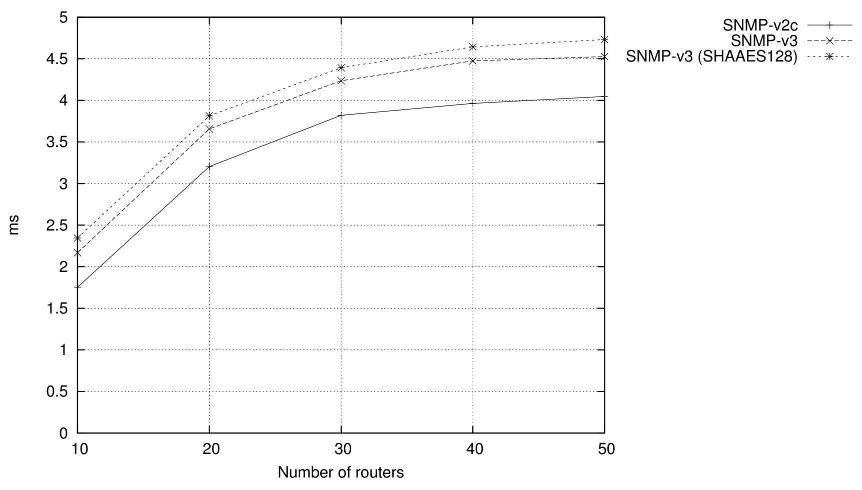


Figure 9.12: Message exchange delay

the network, since the probability is higher that both source and destination of the messages are in the same connected component. If source and destination are not in the same connected component, management of the destination router is not possible from the management station. There is no significant difference of delivery ratio between the different SNMP variants.

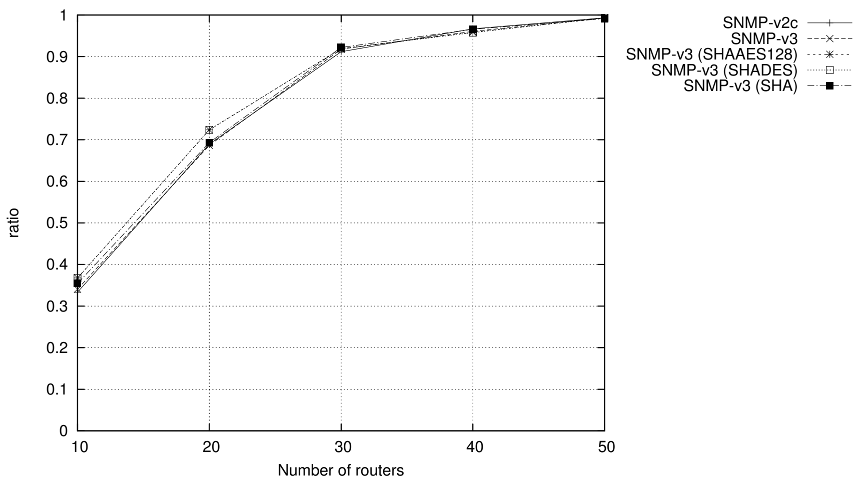


Figure 9.13: Delivery ratio of SNMP messages

Figure 9.14 depicts the delivery ratio when routers are mobile. The delivery ratio decreases, but remains at a relatively high level, due to the low number of collisions and the relatively short distance in hops from the manager to all agents.

Finally, figure 9.15 illustrates the average path length in hops between the SNMP manager and the agents. There is, of course, no significant difference between the different SNMP variants. As the simulation area is constant, and only the number of routers increase (*i.e.* the density increases), the path length converges with increasing number of routers.

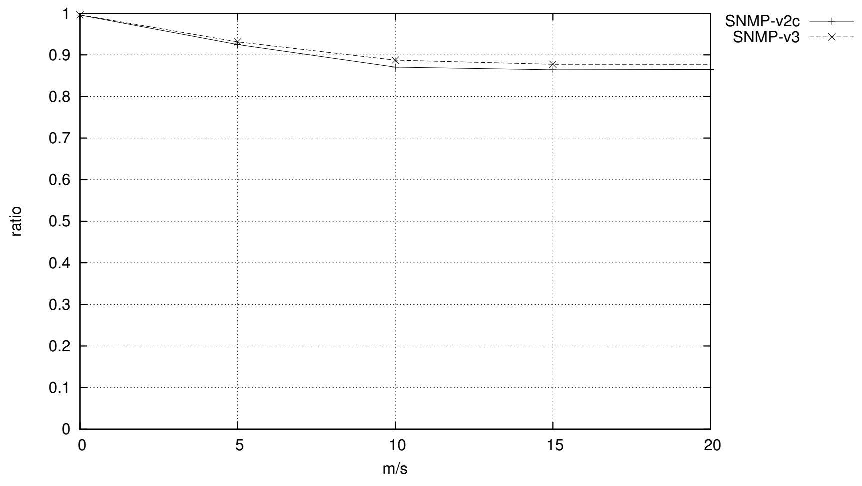


Figure 9.14: Delivery ratio in a mobile network of 50 routers

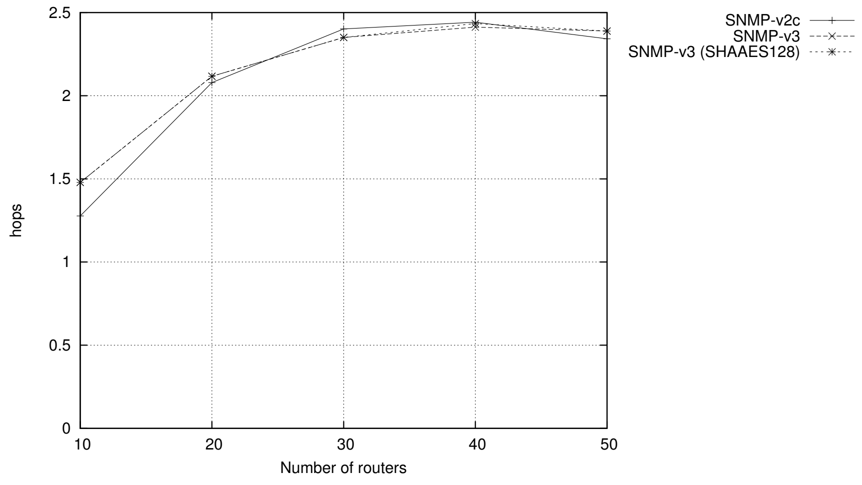


Figure 9.15: Average path length

9.10 Conclusion

OLSRv2 does not require any external interaction once deployed, as routers are able to accommodate frequently changing network topologies in a self-organizing manner, as well as to accommodate OLSRv2 routers with heterogeneous configuration in the same network. However, it is often desirable to monitor the network performance and to "tweak" parameters for improving the performance of an existing deployment of the routing protocol, such as to optimized the performance of the protocol, and therefore to facilitate scalability of the network.

This chapter has proposed a management and monitoring architecture for OLSRv2 routers based on SNMP, which allows a Network Management System (possibly used by a human or automated network operator) (i) to acquire the state of the router (*i.e.* all parameters and information bases of the routing protocol), (ii) to modify parameters during runtime, and (iii) to generate offline performance reports. As for (i) and (ii), two Management Information Bases (MIBs) are proposed for OLSRv2 and for the neighborhood discovery part of OLSRv2, called NHDP. (iii) is derived through the creation of an external proxy service, the REPORT-MIB, located in close proximity of the managed devices where it may poll for the current values necessary for generation of the performance reporting. The rationale for this proxy service is to avoid frequent polling over the network, leading to a frequent and bandwidth-consuming message exchange.

The REPORT-MIB does polling of counters from the OLSRv2-MIB and NHDP-MIB, and creates history-based performance reports based on these counters over time – reports which, then, are made available via SNMP. This chapter specifies a number of such performance reports that concern the stability of the nearby topology of a router. When some of the router parameters in OLSRv2 and NHDP (such as the link quality related parameters) are unwisely set with respect to the characteristics of a given network, the local topology may "flap" between several possible configurations, thus leading to additional control traffic overhead, in-router calculations and deteriorated performance. Detecting such behavior, on a global level and for multiple routers in the same region, could enable appropriately "tuning" the parameters towards a more stable routing structure in the network.

This chapter has also evaluated the behavior and the performance of SNMP in MANETs where routers run OLSRv2. Different evaluation metrics are considered, such as delivery ratio, delay, overhead, collisions, both in static and in mobile networks, with increasing number of routers. Different variants of SNMP, notably SNMPv2c, SNMPv3 without authentication or privacy, SNMPv3 with SHA authentication only, and SNMPv3 with authentication and privacy (AES128 and DES) have been considered.

The contributions of this chapter have been published in [48] and in IETF Working Group drafts [49, 50].

Chapter 10

Delay Tolerant Routing with OLSRv2

Typically, successful IP datagram forwarding on a router depends on whether the routing table of the operating system contains a “valid route” towards the destination of the datagram. If such a valid route exists, the datagram is retransmitted, otherwise, it is dropped. In ad hoc networks, that behavior may lead to low delivery ratio of datagrams if the network topology is “unstable”, *i.e.* if due to the wireless characteristics of the communication media, links between routers break down frequently. In particular in large-scale deployments, it can be observed that routes may be “invalid”, either because of a higher probability of interrupted paths (as each additional hop over a wireless medium increases the probability of packet loss), or due to the higher required convergence time of the routing protocol. In some deployments, it may be preferable to accept a higher delay of datagram delivery by retransmitting the data later when a valid route exists again to the destination, instead of dropping it (“delay tolerant networking”).

This chapter analyzes such delay tolerant behavior in OLSRv2-routed MANETs by means of network simulations in increasingly large MANET deployments. In order to test the effect of delay tolerant routing in “extreme” cases, a network model is proposed, in which routers switch between two states, “disabled” and “enabled”. During the *disabled* state, a router cannot receive or transmit messages. This network mobility model for network simulators, specified in this chapter, leads to a weak performance of routing protocols without delay tolerant routing, whereas high delivery ratios can be achieved – at expense of a higher delay – if datagrams are buffered and scheduled for later transmission.

10.1 Chapter Outline

Section 10.2 specifies a modification of the traditional IP datagram forwarding process for allowing for delay tolerant message forwarding. In order to test that delay-tolerance mechanism under “extreme” conditions in network simulators, a mobility model is proposed in section 10.3. Section 10.4 presents a performance evaluation of the delay-tolerance mechanism in OLSRv2-routed networks simulated with the proposed mobility model. Section 10.5 concludes the chapter.

10.2 Link Buffer Mechanism

This section specifies a link buffer mechanism, based on [148], which allows to delay a datagram transmission in cases where the destination is (temporarily) unavailable, and to transmit the datagram at a later point of time when the destination is available again.

10.2.1 IP Buffering

Traditional IP datagram transmission considers, essentially, two "states" for any destination: either a route exists (in which case IP datagrams can be transmitted) or no route exists (in which case IP datagrams are to be dropped). The proposed link buffering mechanism changes the default behavior of dropping IP datagrams if no route exists to the destination and instead allows to buffer datagrams for expected later delivery. Table 10.1 lists the different states, combined with the appropriate action to be taken.

Table 10.1: Link buffering states and actions

State	**Description**	**Action**
No Route	No routing information is available for this destination	Buffer IP Datagrams
Route Valid	A route entry exists for this destination, and the link to the next hop towards this destination is not disconnected	Transmit IP Datagrams

Whenever the routing protocol adds or modifies a route to a destination, the buffer is notified, and all buffered IP datagrams to that destination are retransmitted.

10.2.2 L2 Buffering

A variant to the mechanism described in section 10.2.1 is to consider a third state in addition to "no route" and "route valid", called "route invalid". This state corresponds to the situation where a destination is believed to be present in the network, although the previously selected "next hop" is currently unavailable. This variant requires the link layer (L2) to be aware if transmissions to the next hop fail, *i.e.* the link to the next hop is disconnected. For example, 802.11 [149] applies such a mechanism using acknowledgments: if a frame is not acknowledged by the destination, the frame is dropped (after a bounded number of failed attempts with an exponential back-off mechanism). If such a mechanism of the link layer is in place, then IP datagrams can also be buffered in the "route invalid" state.

10.2.3 Flow Chart

Figure 10.1 depicts the process of link buffering, in case when both link buffering at routing level (as specified in section 10.2.1) and at link layer (as specified in section 10.2.2) is applied. When a datagram is to be forwarded by a router (state "Datagram to be forwarded"), the router verifies whether it has a route towards the destination. In that case, it hands the datagram down to the lower layers (states "Send datagram" and "Datagram arrives at L2"). If the datagram cannot be sent successfully (*e.g.* no ACK has been received), the datagram is buffered.

If no valid route exists (state "valid route available?"), the behavior depends on whether the router is the source of the datagram. In that case, the datagram is buffered, otherwise it is dropped. This differentiation is further discussed in section 10.2.4.

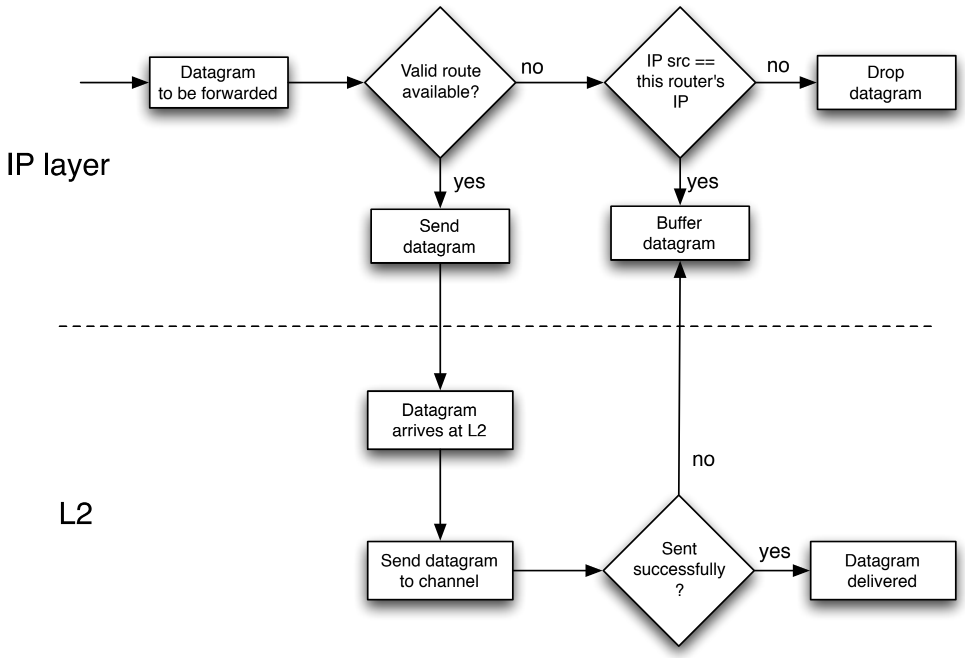


Figure 10.1: Link buffering process for L3 and L2

10.2.4 Buffering at Intermediate Hops

Datagrams may traverse several hops (*i.e.* several routers) from the source to the destination. Invalid routes can occur at all of these routers because routing protocols may take some time until they convergence, *i.e.* until they are "aware" of the effective topology. Typically, the further a router is away from a destination (in terms of hops), the longer is the convergence time for that destination[1]. Thus, it is possible that a router forwarding an IP datagram still has a valid

[1]Unless there are unstable links to a "close" destination in terms of hop count, and stable links to a destination far away

route to the destination and forwards it, while another router further along the path towards the destination has a fresher information about that destination and is aware that the destination is not available any more.

Consequently, buffering could be performed at any of these “intermediate” routers along the path of an IP datagram, and not only at the source. Buffering datagrams at intermediate routers may lead to a large demand of memory on some routers that are bottlenecks in the network. Consider the example in figure 10.2.

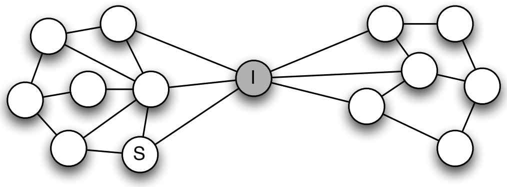


Figure 10.2: The gray router buffers datagrams that it forwards from the left side to the right side of the network (and vice verse)

In the example, the gray router (called ‘I’) buffers datagrams which it forwards from other routers to unavailable destinations. This may lead to buffer overflows on ‘I’, and also exposes the router to denial of service attacks (for example, if the originating router ‘S’ sends many datagrams to ‘I’, destined to a non-available router).

While it may be undesirable for these reasons to buffer datagrams at routers other than the “source” router, it will be considered in the performance evaluation in section 10.4.

10.3 Model Description

In order to test the link buffering mechanism presented in section 10.2 in extreme conditions (*i.e.* where the end-to-end delivery ratio of IP datagrams is otherwise low without such a mechanism), a mobility model for network simulators is presented in this section, called “popup model”. The popup mobility model is defined as follows: Every router in a MANET has two conjunct states (as depicted in figure 10.3): *enabled* and *disabled*.

Whenever a router is “enabled”, it operates as usual, *i.e.* it may exchange control messages of a routing protocol, send and receive data traffic etc. In this state, the router does not move. The router remains for a certain time (denoted `enabled_time`) in this state. It then switches to the “disabled” state which may involve a change in position: the router can move up to a certain upper bounded distance (denoted `max_distance`) with infinite speed.

If a router is “disabled”, it cannot send or receive any messages from the network (neither data nor control traffic). The router can be considered as being “switched off” and is thus completely unresponsive to any network events. The router stays a maximum time of `disabled_time` in that state until it goes back to the “enabled” state. Note that both `enabled_time` and `disabled_time` can be infinite, meaning that a router can remain in any of both states forever.

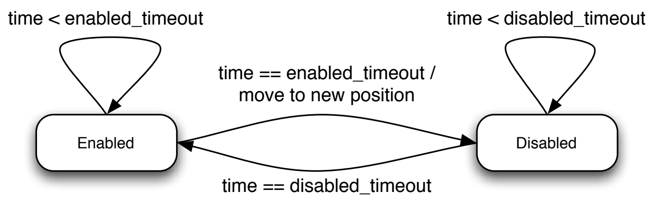
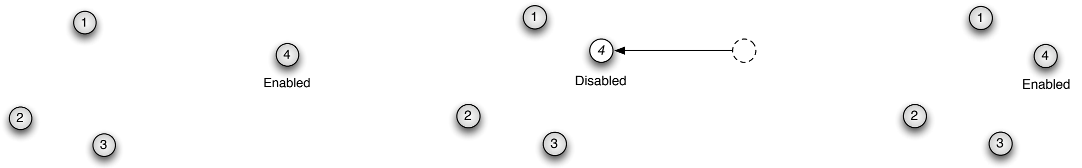


Figure 10.3: State diagram of the popup model

Figure 10.4 depicts an exemplary MANET that behaves according to the above-mentioned popup mobility model. At time t_0, routers 1 to 4 do not move and are all enabled which means they operate as expected, possibly exchanging control and data traffic. In the figure in the middle at time t_1, router 4 becomes disabled and moves with infinite speed up to a certain distance from its original position. At time t_2, depicted in the right figure, the router is enabled again and participates normally in the network again.



(a) At time t_0, routers 1 to 4 do not move and are all enabled

(b) At time t_1, router 4 becomes disabled and moves

(c) At time t_2, router 4 is enabled again, not moving any more

Figure 10.4: Example of the popup mobility model

10.4 Performance Evaluation

This section presents a performance evaluation of unicast data traffic in OLSRv2-routed MANETs, both *with* and *without* the link buffering mechanism described in section 10.2, using NS2. In the evaluation, typical metrics such as delivery ratio of data traffic, control traffic overhead, average path length and delay of data traffic are considered.

10.4.1 God Routing Protocol

In order to provide an upper bound in terms of delivery ratio for the simulated scenarios, a "God routing protocol" (denoted "GodRP") has been implemented. The routing protocol uses the "God" object of NS2, in order to calculate routes to all destination based on their position and radio range, without any control message exchange and zero convergence time. The expected performance of the GodRP is close to the best possible routing protocol, which helps to understand how well a routing protocol could theoretically perform[2].

A variation of the GodRP is included in the simulation as well, which performs link buffering as described in section 10.2, denoted "GodRP-LB". It has to be noted that contrary to GodRP, the variant with link buffering (LB), does not represent an upper bound in terms of data delivery ratio, which shall be illustrated in the example in figure 10.5. Four routers run a link state routing

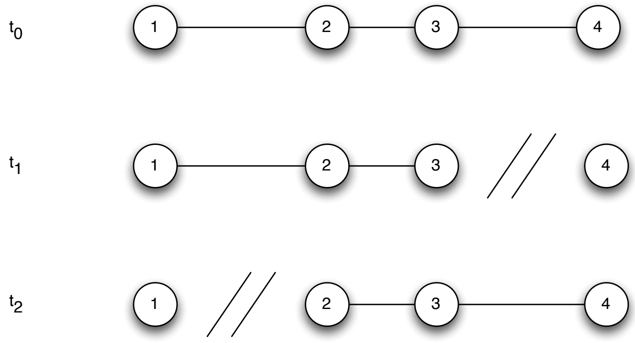


Figure 10.5: GodRP with link buffering ("GodRP-LB") does not represent an upper bound in terms of data delivery ratio for routing protocols with link buffering

protocol and are arranged in a "line" topology at time t_0. Router 1 sends unicast data traffic to router 4. At time t_1, the link between router 3 and 4 breaks. Due to the non-zero convergence time of link state routing protocols, router 1 may still have a valid route to 4, so it transmits the datagrams. When router 3 receives the datagrams (after they have been forwarded by router 2), it may have a fresher information about the disconnected link to 4. If no intermediate buffering is used, it will drop the datagram. However, when using intermediate buffering, the datagram is buffered, and delivered later (at time t_2). Using GodRP-LB, the datagram will never be delivered in this example from t_1 on: At t_1, router 1 will be immediately informed about the link disconnection between 3 and 4, so router 1 will buffer all datagrams. But since at t_2, router 1 is not connected any more to 2, datagrams will not reach their final destination, router 4.

10.4.2 Graceful Shutdown

In the performance evaluation, a variation of OLSRv2 with link buffering as considered: if routers are gracefully shutdown when transiting from "enabled" to the "disabled" state, they send a HELLO to inform neighbors to consider the link between them as unavailable. The example in figure 10.6 depicts the graceful shutdown process. At time t_0, all three routers are in the "enabled" state. In the periodic HELLO messages from router 2, it will list both router 1 and 3 as symmetric (SYM) neighbors. Thus, router 1 will list 2 as symmetric neighbor, and 3 as two-hop neighbor reachable via 2. When router 2 is about to be shut down at time t_1, it sends a HELLO message with all neighbors listed as LOST. Thus, router 1 will remove the 2-hop

[2]In some cases, a perfect routing protocol would perform better to drop some packets even if they could be routed, *e.g.* to destinations further away in cases of massive load on the channel. This may lead to a higher delivery ratio for destinations close to the router (*i.e.* neighbors) if the transmissions contend for the channel.

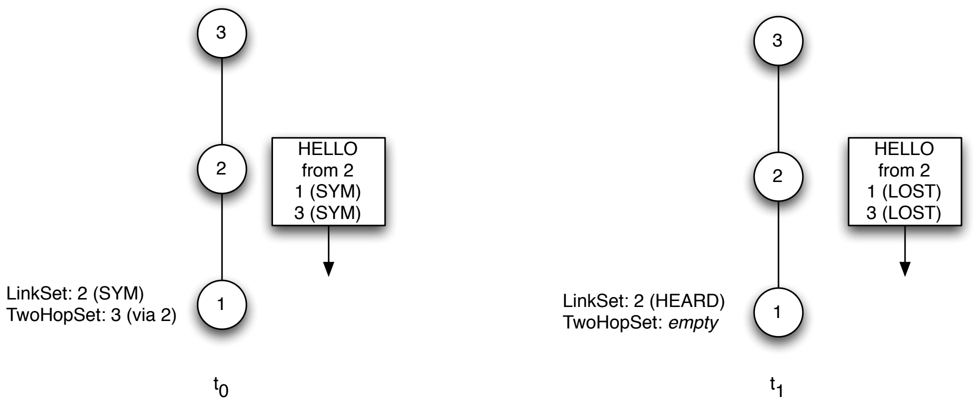


Figure 10.6: Graceful shutdown

neighbor entry, and set the link status towards router 2 to HEARD. In addition, router 1 may trigger a HELLO message (and possibly a TC message, if it was selected as MPR by router 2), advertising the lost link.

Without the graceful shutdown mechanism, router 1 would have listed 2 as symmetric neighbor until the link has expired. Thus, the graceful shutdown mechanism effectively reduces the convergence time of the routing protocol.

10.4.3 Simulation Settings

The settings for the performed NS2 simulations are presented in table 10.2. The routers behave according to the popup mobility model as defined in section 10.3 in a square area of 1000 meters length. In average, there are five data streams with a constant bit rate of 3.2 kB/s between any two random routers. The simulation time is 300 seconds, and each data point was averaged over 30 runs. A random seed was used for every simulation. The following routing protocol variants are compared: OLSRv2, OLSRv2 with link buffering on L3 only ("OLSRv2-L3-LB"), OLSRv2 with link buffering ("OLSRv2-LB"), OLSRv2 with link buffering and graceful shutdown ("OLSRv2-LB-gf"), OLSRv2 with link buffering also at intermediate routers ("OLSRv2-IM-LB"), the God routing protocol ("GodRP"), God routing protocol with link buffering ("GodRP-LB").

At the beginning of the simulation, all routers are set in either "disabled" or "enabled" state randomly with a random `enabled_time` or `disabled_time` left before the next state transition.

10.4.4 Simulation Results

Figure 10.7 shows the unicast data delivery ratio. The God routing protocol – not surprisingly – has a higher delivery rate than OLSRv2, which is due to the zero-convergence time and less collisions due to the lack of control traffic. However, even with the GodRP, not more than 35% of delivery ratio with 70 routers is reached. The link buffering variants lead to much higher delivery ratio, with GodRP with link buffering having the highest rate. OLSRv2 with intermediate link

Table 10.2: Simulation settings

Parameter	Value
NS2 version	2.34
Mobility model	Popup model
Area	1000m x 1000m
Number of routers	20 - 70
Communication range	250m
Radio propagation model	Two-ray ground
Enabled time	25 – 30s
Disabled time	25 – 30s
Move distance	100m
Simulation time	300s
Avg. number of concurrent CBR streams	5
Interface type	802.11b
Radio frequency	2.4 GHz
OLSRv2 parameters	Proposed default values of [39]

buffering is not far below the GodRP-LB. Due to the non-zero convergence time of OLSRv2, buffered datagrams will be delivered only a bit later than with the link buffering variant of the GodRP (which can be observed in figure 10.8). OLSRv2 without intermediate link buffering has a significantly lower delivery ratio, whereas the graceful shutdown mechanism has a positive effect on the delivery ratio due to the reduced convergence time.

Figure 10.8 depicts the average delay from the moment a datagram has been sent until it arrives at its destination. The delay largely depends on the selected `enabled_time` and `disabled_time` values. All buffering variants have a significantly higher delay than standard OLSRv2 (at the benefit of a higher delivery ratio). The different buffering variants lead to similarly high delay, however, the delay of OLSRv2-IM-LB is significantly higher than the other protocols. This can be explained with the increased routing stretch – paths towards the destination are longer, as observed in figure 10.9.

Figure 10.9 presents the average path length of every data packet from its source to its destination. OLSRv2 has the shortest paths; this can be explained by the lowest delivery ratio amongst the compared protocols (as depicted in figure 10.7) – destinations further away from the source receive fewer of the transmitted datagrams. For the same reason, the other link buffering variants have a lower path length than GodRP-LB. Comparing GodRP-LB and OLSRv2-IM-LB (which both have very similar delivery ratios as shown in figure 10.7), the path stretch of OLSRv2-IM-LB can be clearly observed.

Figure 10.10 shows the accumulated control traffic overhead over the simulation time (GodRP is not included, because the overhead is 0). Not surprisingly, the different link buffering variants of OLSRv2 have no influence on the control traffic. The graceful shutdown mechanism includes transmitting additional HELLOs before shutdown of a router, plus possibly triggered HELLOs and TCs of neighbors of that router.

Figure 10.11 shows the total number of dropped frames due to collisions during the simulation. For GodRP, almost no collisions appear, since no control traffic congests the network and only

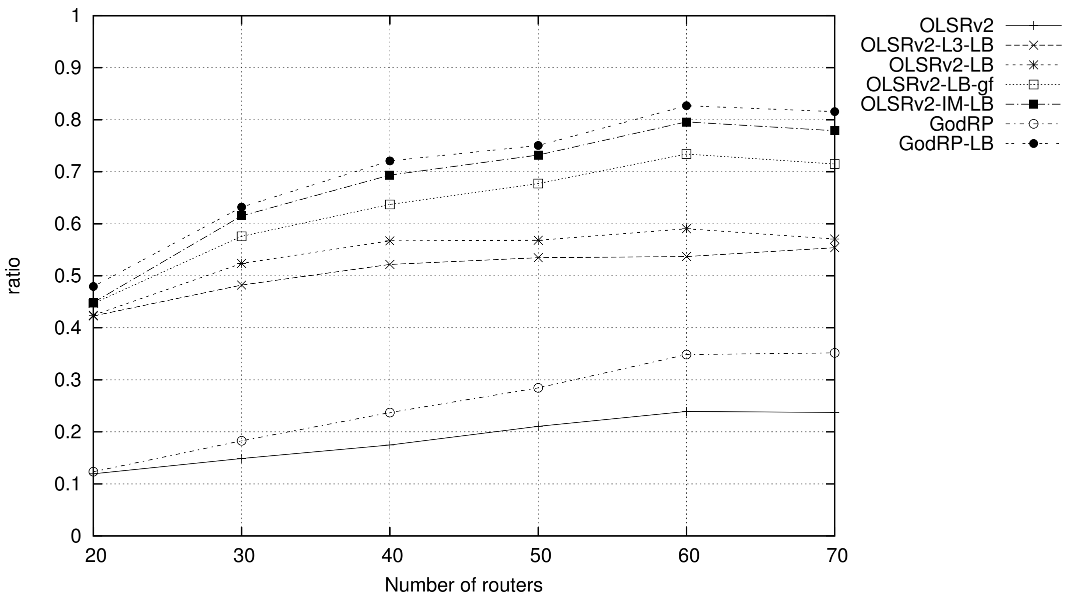


Figure 10.7: Unicast data delivery ratio

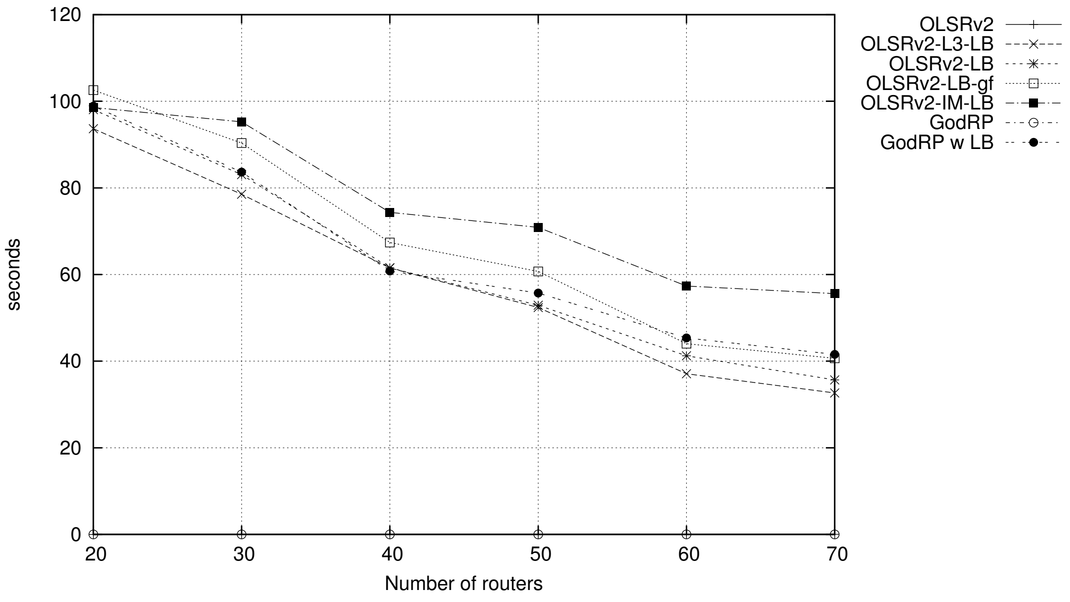


Figure 10.8: Average delay per datagram

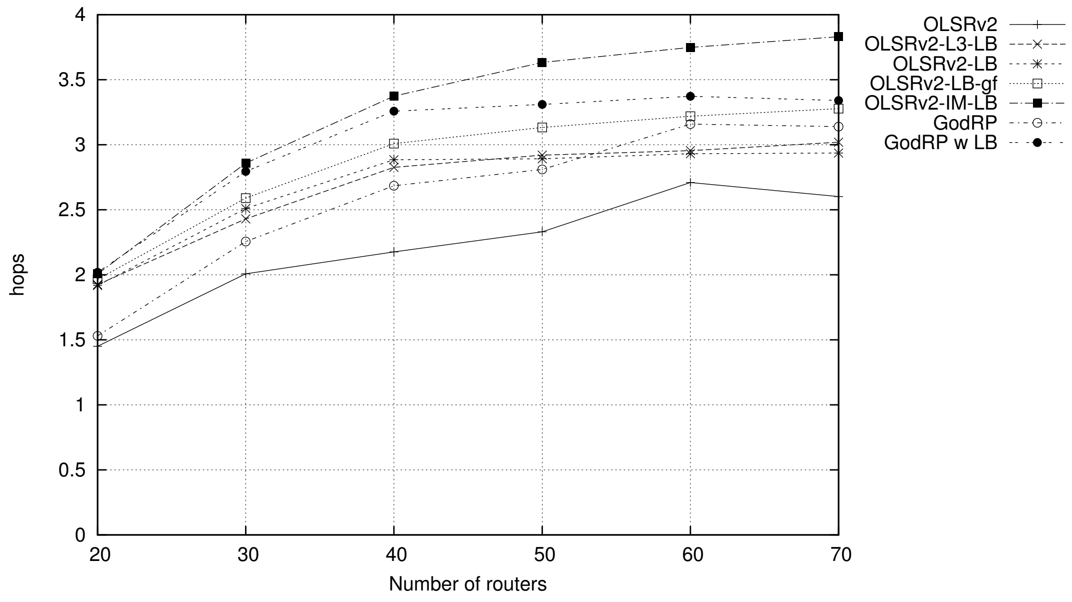


Figure 10.9: Path length

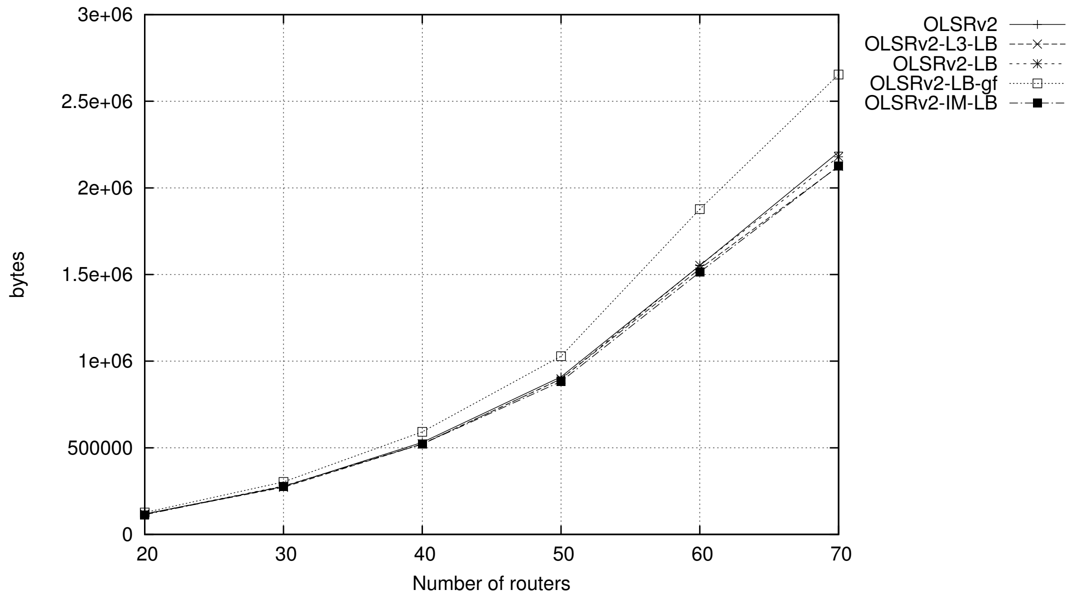


Figure 10.10: Accumulated control traffic overhead

little unicast traffic is injected in the network. OLSRv2 without link buffering has the lowest number of collisions, increasing with a growing number of routers in the network – the increasing amount of control traffic leads to most of the collisions. All link buffering variants have a higher collision ratio, which can be explained by the fact that once a router reappears (*i.e.* changes its state from "disabled" to "enabled"), all datagrams from different sources that have been buffered while it was "disabled", are retransmitted almost the same time, as soon as valid routes towards that destination router are added to the routers that had buffered the datagrams. Again, collisions due to increasing control traffic explain the difference between the link buffering variants of OLSRv2 and GodRP-LB with higher number of routers. The bottleneck effect, that has been described in section 10.2.4, can be well observed due to the large number of collision when applying intermediate buffering (OLSRv2-IM-LB).

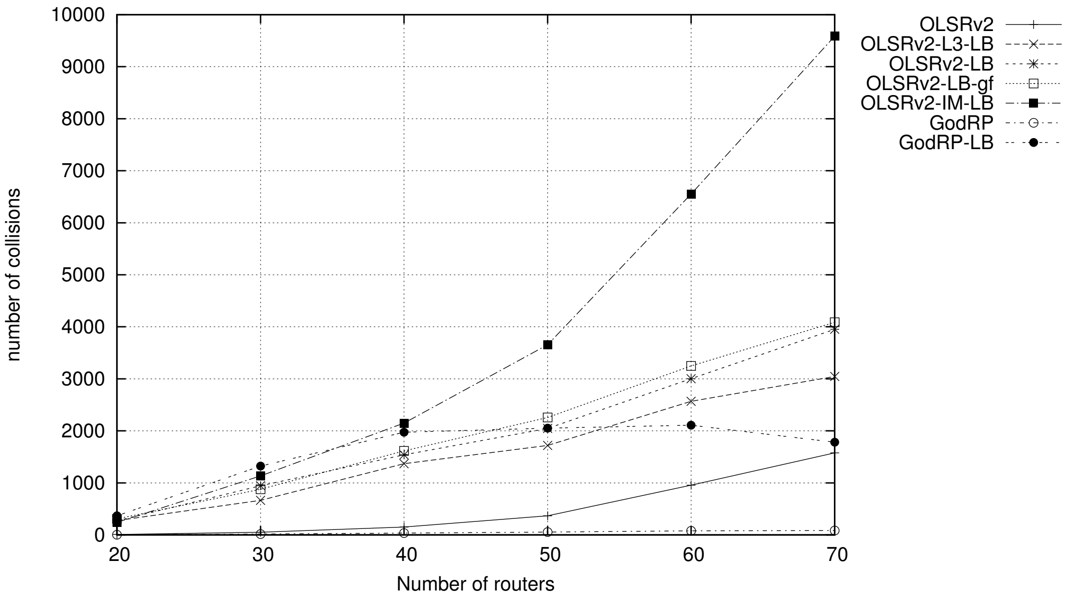


Figure 10.11: Collisions

The fact that despite the high number of collision with OLSRv2-IM-LB, the delivery ratio (as depicted in figure 10.7) is still the highest amongst the OLSRv2 variants, can be explained by the retransmission mechanism of 802.11 [149]. Even if more frames are lost due collisions on L2, eventually one may reach the destination if the frame (*i.e.* the next hop along the route). Figure 10.12 depicts the total accumulated overhead of unicast data traffic, when counting each L2 (re-)transmission at every router along the path from source to destination.

It can be seen that OLSRv2-IM-LB has the highest data overhead: The higher delivery ratio despite the higher collision ratio comes at the expense of available bandwidth on the channel.

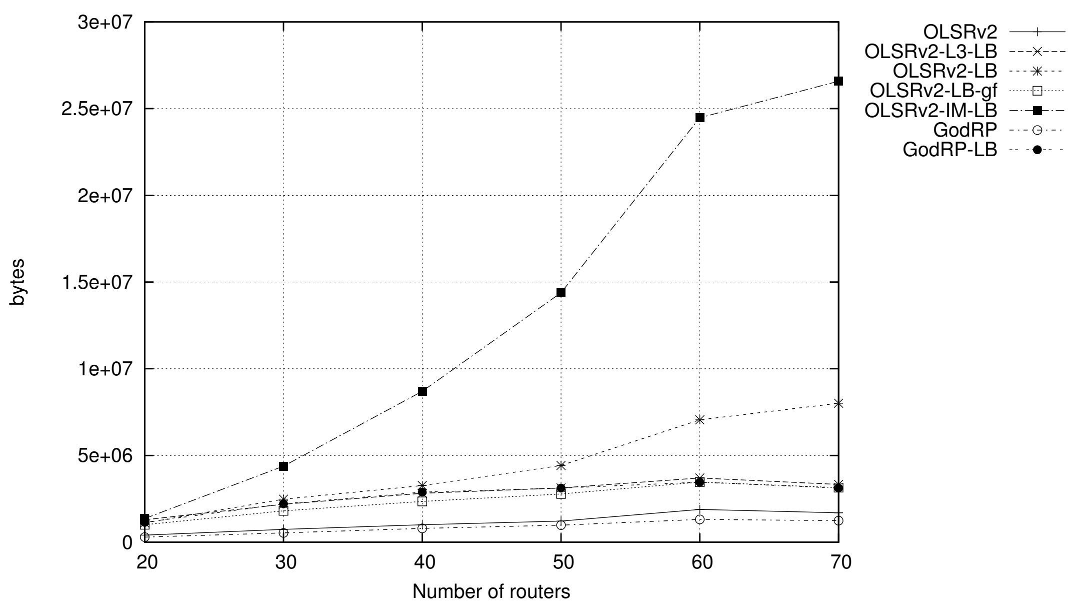


Figure 10.12: Accumulated unicast traffic overhead (counting each transmission at every router)

10.5 Conclusion

This chapter has proposed a “link buffering” mechanism allowing a router to delay the transmission of an IP datagram if there is no valid route to the destination of that datagram at transmission time. In large, “unstable” network deployments, where wireless links break down frequently, such a mechanism allows to increase end-to-end delivery ratio of datagrams, at the expense of higher delay. In order to test that mechanism in such “unstable” deployments, a mobility model for network simulators, denoted “popup model” is specified: routers can be either “enabled” or “disabled”. In the “disabled” state, routers cannot send or receive any data, but are allowed to change position within a bounded distance.

Traditional IP datagram forwarding (*i.e.* datagrams are dropped when no valid route exists) in OLSRv2-routed MANETs adhering to the popup model leads to a low delivery ratio, as demonstrated in this chapter by NS2 simulations. Even simulations with a “God routing protocol”, *i.e.* a routing protocol with zero convergence time and no control traffic, entail a low delivery ratio.

The simulations have shown that using a “link buffering” mechanism, delivery ratio can be largely increased at expense of a significantly higher end-to-end delay. Several variants of the link buffering mechanism are considered, a variant with buffering on layer 3 only (denoted “OLSRv2-L3-LB”), link buffering on layer 2 and 3 (denoted “OLSRv2-LB”), link buffering on intermediate routers (denoted “OLSRv2-IM-LB”), and a variant with a graceful shutdown mechanism (denoted “OLSRv2-LB-gf”). All link buffering variants entail a much higher end-to-end delivery ratio than without using such a mechanism, but also a much higher delay.

Chapter 11

Vulnerability Analysis of OLSRv2

The developments reflected in OLSRv2 have been motivated by increased real-world deployment experiences, *e.g.* from networks such as FunkFeuer, and the requirements presented for continued successful operation of these networks. With participation in such networks increasing, operating with the assumption, that participants can be "trusted" to behave in a non-destructive way, is utopia. Taking the Internet as an example, as participation in the network increases and becomes more diverse, more efforts are required to preserve the integrity and operation of the network. Most SMTP-servers were, *e.g.*, initially available for use by everyone on the Internet – with an increased populace on the Internet, attacks and abuses caused the recommended practice is today to require authentication and accounting for users of such SMTP servers [150].

In chapter 12, a security mechanism to OLSRv2 is proposed and evaluated. However, a first step towards hardening against attacks disrupting the connectivity of a network, is to understand the vulnerabilities of the routing protocol, before providing a mechanism that protects the routing protocol against (part of) these vulnerabilities. This chapter therefore analyzes OLSRv2, to understand its inherent vulnerabilities and resilience. This analysis is not claimed to be complete, but may form a meaningful starting-point for developing secured OLSRv2 networks.

11.1 Chapter Outline

Section 11.2 presents related work to this chapter. Section 11.3 provides an overview of base assumptions of OLSRv2, as well as attack vectors, that are used in the following detailed description of attacks against OLSRv2. Section 11.4, 11.5, 11.6 and 11.7 each represents a class of disruptive attacks against OLSRv2, detailing a number of attacks in each class. Section 11.8 summarizes inherent resilience, as observed in OLSRv2. The chapter is concluded in section 11.9.

11.2 Related Work

A number of articles have analyzed security properties and vulnerabilities of routing protocols in MANETs ([151, 152, 153, 154]). These papers identify resources of MANET routing protocols that are potentially vulnerable to attacks, and propose several attacks against these resources, as well as counter-measures against such attacks.

[155, 156, 157] present a security analysis of the OLSR routing protocol (in its first version as specified in [79]). The work described in this chapter presents a more detailed analysis of the successor of OLSR, by taking an abstract look at each of the algorithms that constitute OLSRv2 and by identifying for each protocol element the possible vulnerabilities and attacks. Moreover, while the main basic algorithms of OLSR have been integrated into OLSRv2, some of the presented vulnerabilities in this chapter are specific to OLSRv2.

11.3 Overview

This section provides an overview of base assumptions of OLSRv2 and link state protocols in general, as well as attack vectors against OLSRv2.

11.3.1 Link State Vulnerability Taxonomy

Proper functioning of OLSRv2 assumes that (i) each router can acquire and maintain a topology map, accurately reflecting the effective network topology; and (ii) that the network converges, *i.e.* that all routers in the network will have sufficiently identical topology maps. An OLSRv2 network can be disrupted by breaking either of these assumptions, specifically (a) routers may be prevented from acquiring a topology map of the network; (b) routers may acquire a topology map which does not reflect the effective network topology; and (c) two or more routers may acquire inconsistent topology maps.

11.3.2 OLSRv2 Attack Vectors

Besides “radio jamming”, attacks on OLSRv2 consist of a malicious router injecting “correctly looking, but invalid, control traffic” (TCs, HELLOs) into the network. A malicious router can either (a) lie about itself (its ID, its willingness to serve as MPR), henceforth *Identity Spoofing* or (b) lie about its relationship to other routers (pretend existence of links to other routers), henceforth *Link Spoofing.* Such attacks will eventually cause disruption in the Link State Advertisement process, through targeting the MPR Flooding mechanism, or by causing incorrect link state information to be included in TCs, causing routers to have incomplete, inaccurate or inconsistent topology maps. In a different class of attacks, a malicious router injects control traffic, tuned to cause an *in-router resource exhaustion*, *e.g.* by causing the algorithms calculating routing tables or MPR sets to be invoked continuously, preventing the internal state of the router from converging.

11.4 Topology Map Acquisition

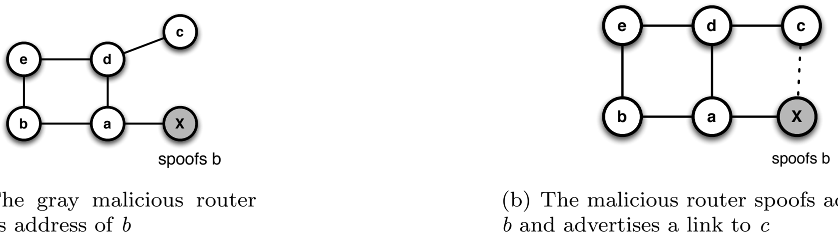
Topology Map Acquisition relates to the ability for any given router in the network to acquire a representation of the network connectivity. A router, unable to acquire a topology map, is incapable of calculating routing paths and participating in forwarding data. Topology map acquisition can be hindered by (i) TCs to not being delivered to (all) routers in the network, such as what happens in case of Flooding Disruption, or (ii) in case of "jamming" of the communication channel.

11.4.1 Flooding Disruption

MPR selection (section 6.3.2) uses information about a router's 1-hop and 2-hop neighborhood, assuming that (i) this information is accurate, and (ii) all 1-hop neighbors are equally apt as MPR. Thus, a malicious router seeking to attack the MPR Flooding process will seek to manipulate the 1-hop and 2-hop neighborhood information in a router such as to cause the MPR selection to fail.

11.4.1.1 Flooding Disruption due to Identity Spoofing

Figure 11.1(a) illustrates a network in which the malicious router (gray circle) spoofs the identity of b, *i.e.* a receives HELLOs from two routers, both pretending to be b. As HELLOs are additive, and with the malicious router X not advertising any neighbors, the topological view of the 1-hop and 2-hop neighborhood of a is unaffected by the presence of X: a's MPR selection will function correctly by selecting (if using a greedy algorithm) d.



(a) The gray malicious router spoofs address of b

(b) The malicious router spoofs address of b and advertises a link to c

Figure 11.1: Identity Spoofing: flooding attack: 1-hop address duplication.

Figure 11.1(b) illustrates a network in which the malicious router X (gray circle) spoofs the identity of b. In this example, a link (the dotted line) between X and c is correctly detected and advertised by X. Router a will receive HELLOs indicating that links exist from b to both e and c, thereby rendering b a candidate MPR on par with d.

If X does not forward flooded traffic (*i.e.* does not accept MPR selection), its presence entails a flooding disruption: selecting b over d renders c unreachable by flooded traffic. In order to increase the likelihood that the malicious X is selected, it may set its willingness to the maximum value, ensuring that it is always selected and the routers so covered are not further considered in the MPR selection algorithm.

Figure 11.2 illustrates a network in which the malicious router X (gray circle) spoofs the identity of x, *i.e.* a and c both receive HELLOs from a router pretending to be x. From the point of view of b, it appears as if a and c have the same neighbor set, hence either is a suitable

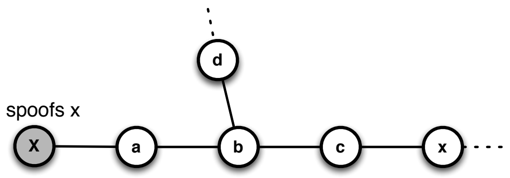


Figure 11.2: Identity Spoofing: flooding attack: 2-hop address duplication.

choice as MPR. Assuming that *b* selects *a* as MPR, *c* will not relay flooded traffic and thus the legitimate (white) *x* (and routers to the "right" of *x*) will not receive flooded traffic.

In order to maximize the impact of the disruption, the malicious router may simultaneously "spoof" multiple identities: by overhearing control traffic for a while, the malicious router may attempt to learn the identities of neighbors of *c* and spoof these – and, in addition, assume one additional identity (possibly not otherwise present in the network). A way of achieving this is to simply have *X* overhear all TCs, and spoof all identities of all routers in the network (possibly excluding *a*). Router *b* will learn through the HELLOs of *a* that all these identities are 2-hop neighbors of *a*. As the set of identities spoofed by the malicious *X* is a superset of the neighbors of *c*, this will cause selection of *a* as MPR, and consequently that *c* is not selected.

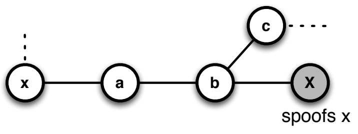


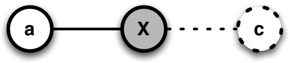
Figure 11.3: Identity Spoofing: flooding attack: 1 and 2-hop address duplication.

Figure 11.3 illustrates a network in which the malicious router *X* (gray circle) spoofs the identity of *x*, *i.e.* *a* and *b* both receive HELLOs from a router pretending to be *x*. Router *b* will therefore not select *a* as MPR as all the 2-hop neighbors reachable via *a* are already reachable directly in one hop. As a consequence, the white *x*, and any routers "to the left" of it, will not receive flooded control traffic from *b* or transited via *b* from, *e.g.*, *c*.

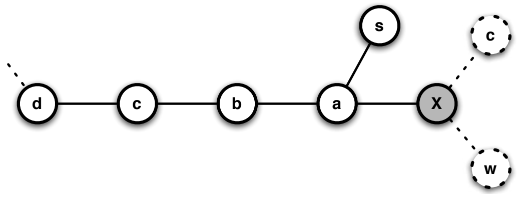
11.4.1.2 Flooding Disruption due to Link Spoofing

Figure 11.4(a) illustrates a network, in which the malicious router *X* spoofs links to the non-existing *c*, *i.e.* *a* receives HELLOs from *X*, pretending the existence of a link between *X* and *c*. This forces *a* to select *X* as MPR – whereas it otherwise would not need to select any MPRs. In this simple example, this does no harm as such.

Figure 11.4(b) illustrates a network, in which the malicious *X* spoofs links to the existing *c*, as well as to a non-existing *w*. Router *a* receives HELLO from *X* reporting links to *c* and *w*, and from *b* reporting a link to *c* only. Unless if *b* has a advertised a maximum willingness, this will cause *a* to select *X* as its only MPR, as *X* presumably covers all 2-hop neighbors of *a* (*i.e.* the real neighbors of *a* as well as the imaginary *w*).



(a) Basic Link Spoofing effect on MPR Selection



(b) Flooding disruption due to Link Spoofing

Figure 11.4: Link Spoofing: Flooding Disruption

The consequence is that as a will not select b as MPR, b will not relay flooded messages received from a. Thus, the network to the left of b (starting with c) will not receive any flooded messages from or transiting a, such as a message originating from s and transiting through a.

11.4.2 Radio Jamming

Radio jamming is an attack in which legitimate access to the communications channel between routers is forcefully hindered by a malicious device. The classic example hereof is where a powerful transmitter is generating "white noise" over the communications channel where the network interfaces of the routers would otherwise operate, effectively preventing these router interfaces from successfully receiving transmissions from each other. While this can happen on all network interface and channel types, wireless networks are especially vulnerable to such; commercial WiFi "jammers" are, for example, readily available [158].

The consequence of such jamming is that the router interfaces, which are so "jammed", are unable to receive routing protocol control traffic, and so are unable to participate in the network. A router where all its network interfaces are victim to "jamming" is, effectively, unable to acquire a topology map of the network and, so, is disconnected from the network.

It can be observed that a router with multiple network interfaces accessing different communications channels, and where not all communications channels are jammed, may still be able to participate in a network via links over these non-jammed interfaces.

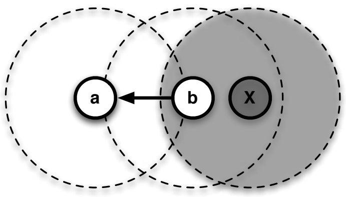


Figure 11.5: Radio Jamming: X is jamming reception within the grayed-out area, thus b is unable to correctly receive transmission.

It can also be observed that while direct jamming affects reception, it may (depending on which lower layers L1/L2 are employed) not affect transmission. Thus, and as illustrated in figure 11.5, *a*, may receive transmissions from *b*, the latter of which is otherwise "jammed" by *X*, which prevents receptions in the grayed area.

The Neighborhood Discovery mechanism of OLSRv2 identifies uni- and bi-directionality of links, and only bi-directional links are advertised and used for routing path calculations. OLSRv2 has, thus, by virtue of this detection and use of only bi-directional links, some resilience to jamming: while the jammed routers are unable to acquire and maintain a topology map of the network, the jammed routers appear as simply "disconnected" to the un-jammed part of the network – which is able to both maintain accurate and consistent topology maps.

11.4.3 Attack on Jittering

OLSRv2 incorporates a jittering: a random, but bounded, delay on outgoing control traffic. This may be necessary when link layers (such as 802.11 [149]) are used, which do not guarantee collision-free delivery of frames, and where jitter can reduce the probability of collisions of frames on lower layers [34].

In OLSRv2, TC forwarding is jittered by a value between 0 and `MAX_JITTER`. In figure 11.6, a router receives three packets, each containing one TC to be forwarded. For each of these, the scheduled retransmission time is calculated as "now plus jitter", illustrated by the horizontal arrows.

In order to reduce the number of transmissions, when a control message is due for transmission, OLSRv2 piggybags all queued messages into a single transmission. Thus, if a malicious router sends many TCs within a very short time interval, the jitter time of the attacked router tends to 0. This renders jittering ineffective and can lead to collisions on L2.

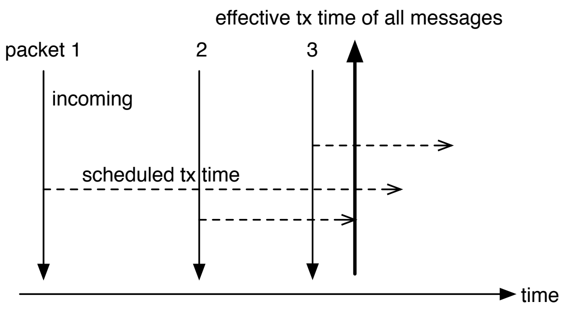


Figure 11.6: Jittering: If several messages are scheduled to be transmitted, all the messages are sent at the minimum of the scheduled transmission times.

11.4.4 Hop Count and Hop Limit Attacks

The hop-count and hop-limit fields are the only parts of a TC that are modified when forwarding. A malicious router can modify either of these when, when forwarding TCs.

11.4.4.1 Modifying the Hop Limit

A malicious router can decrease the hop limit when forwarding a TC. This will reduce the scope of forwarding the message, and may lead to some routers in the network not receiving that TC. Note that this is not necessarily the same as *not* relaying the message (*i.e.* setting the hop limit to 0), as illustrated in figure 11.7.

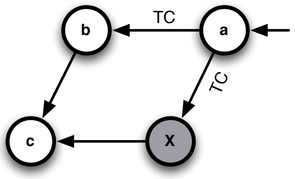


Figure 11.7: Hop limit attack

A TC arrives at and is forwarded by *a*, such that it is received by both *b* and the malicious *X*. It is assumed that router *a* has selected both router *b* and *X* as MPRs, *i.e.* that they are both supposed to forward the TC message. *X* can forward the TC without any delay (including without jitter) such that its transmissions arrives before that of *b* at *c*. Before forwarding, it significantly reduces the hop limit of the message. Router *c* receives the TC, processes (and forwards) it, and marks it as already received – causing it to discard further copies received from *b*. Thus, if the TC is forwarded by *c*, it has a low hop limit and may not reach all routers in the network.

11.4.4.2 Modifying the Hop Count

A malicious router can modify the hop count when forwarding a TC. This may have two consequences: (i) if the hop count is set to the maximum value, then the TC will be forwarded no further by recepients, or (ii) artificially manipulating the hop count may affect the validity time as calculated by recipients, when using distance-dependent validity times as defined in [36] (*e.g.* as part of a fish-eye extension to OLSRv2 [159]).

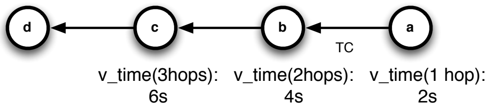


Figure 11.8: Different validity times based on the distance in hops

In figure 11.8, *a* sends a TC with a validity time of two seconds for neighbors that are one hop away, four seconds for routers in a two-hop distance and six seconds in a three-hop distance. If *c* is a malicious router and modifies the hop count (say, by decreasing it to 0), then *d* will calculate the validity time of received information to two seconds – after which it expires unless refreshed. If TCs from *a* are sent less frequently than that up to 3 hops, this causes links advertised in such TCs to be only intermittently available to *d*.

11.5 Effective Topology

Link-state protocols assume that each router can acquire an accurate topology map, reflecting the *effective network topology.* This implies that the routing protocol, through its message exchange, identifies a path from a source to a destination, and this path is valid for forwarding data traffic. If an attacker disturbs the correct protocol behavior, the perceived topology map of a router can permanently differ from the effective topology.

Considering the example in figure 11.9(a), which illustrates the topology map as acquired by s. This topology map indicates that the routing protocol has identified that for s, a path exists to d via b, which it therefore assumes can be used for transmitting data. If, effectively, b does not forward data traffic from s, then the topology map in s does not accurately reflect the effective network topology. Rather, the effective network topology from the point of view of s would be as indicated in figure 11.9(b): d is not part of the network reachable from router s.

(a) Perceived topology by s (b) Effective topology

Figure 11.9: Incorrect Data Traffic Forwarding

11.5.1 Incorrect Forwarding

OLSRv2 routers exchange information using link-local transmissions (link-local multicast or limited broadcast) for their control messages, with the routing process in each router retransmitting received messages destined for network-wide diffusion. Thus, if the operating system in a router is not configured to enable forwarding, this will not affect the operating of the routing protocol, or the topology map acquired by the routing protocol. It will, however, cause a discrepancy between the effective topology and the topology map, as indicated in figure 11.9(a) and figure 11.9(b).

This situation is not hypothetical. A common error seen when deploying OLSRv2 based networks using Linux-based computers as router is to neglect enabling IP forwarding.

11.5.2 Wormholes

A wormhole, depicted in the example in figure 11.10, may be established between two collaborating *devices*, connected by an *out-of-band* channel; these devices send traffic through the "tunnel" to their alter-ego, which "replays" the traffic. Thus, d and s appear as-if direct neighbors and reachable from each other in 1 hop through the tunnel, with the path through the MANET being 100 hops long.

The consequences of such a wormhole in the network depends on the detailed behavior of the wormhole. If the wormhole relays only control traffic, but not data traffic, the same considerations as in section 11.5.1 applies. If, however, the wormhole relays all traffic, control and data alike, it is connectivity-wise identical to a usable link – and the routing protocol will correctly generate a

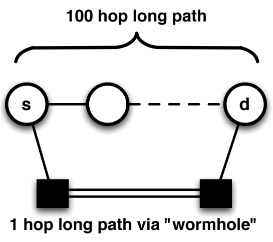


Figure 11.10: Wormholing between two collaborating devices not participating in the routing protocol.

topology map reflecting the effective network topology. The efficiency of the topology so obtained depends on (i) the wormhole characteristics, (ii) how the wormhole presents itself and (iii) how paths are calculated.

Assuming that paths are calculated with unit-cost for all links, including the “link” presented by the wormhole: if the real characteristics of the wormhole are *as-if* it was a path of more than 100 hops (*e.g.* with respect to delay, bandwidth, ...), then the presence of the wormhole results in a degradation in performance as compared to using the non-wormhole path. Conversely, if the “link” presented by the wormhole has better characteristics, the wormhole results in improved performance.

If paths are calculated using non-unit-costs for all links, and if the cost of the “link” presented by the wormhole correctly represents the actual cost (*e.g.* if the cost is established through measurements across the wormhole), then the wormhole may in the worst case cause no degradation in performance, in the best case improve performance by offering a better path. If the cost of the “link” presented by the wormhole is misrepresented, then the same considerations as for unit-cost links apply.

An additional consideration with regards to wormholes is, that such may present topologically attractive paths for the network – however it may be undesirable to have data traffic transit such a path: an attacker could, by virtue of introducing a wormhole, acquire the ability to record and inspect transiting data traffic.

11.5.3 Sequence Number Attacks

OLSRv2 uses two different sequence numbers in TCs, to (i) avoid processing and forwarding the same message more than once (Message Sequence Number), and (ii) to ensure that old information, arriving late due to *e.g.* long paths or other delays, is not allowed to overwrite fresher information (Advertised Neighbor Sequence Number – ANSN).

11.5.3.1 Message Sequence Number

An attack may consist of a malicious router spoofing the identity of another router in the network, and transmitting a large number of TCs, each with different Message Sequence Numbers. Subsequent TCs with the same sequence numbers, originating from the router whose identity was spoofed, would thence be ignored, until eventually information concerning these “spoofed” TCs expires.

11.5.3.2 Advertised Neighbor Sequence Number (ANSN)

An attack may consist of a malicious router spoofing the identity of another router in the network, and transmitting a single TC, with an ANSN significantly larger than that which was last used by the legitimate router. Routers will retain this larger ANSN as “the most fresh information” and discard subsequent TCs with lower sequence numbers as being “old”.

11.5.4 Message Timing Attacks

In OLSRv2, each control message may contain explicit “validity time” and “interval time”, identifying the duration for which information in that control message should be considered valid until discarded, and the time until the next control message of the same type should be expected [36].

11.5.4.1 Interval Time Attack

A use of the expected interval between two successive HELLOs is for determining the link quality in Neighbor Discovery process, as described in [79]: if messages are not received with the expected intervals (*e.g.* a certain fraction of messages are missing), then this may be used to exclude a link from being considered as useful, even if (some) bi-directional communication has been verified. If a malicious X spoofs the identity of an existing a, and sends HELLOs indicating a low interval time, b receiving this HELLO will expect the following HELLO to arrive within the interval time indicated – or otherwise, decrease the link quality for the link a-b. Thus, X may cause b’s estimate of the link quality for the link a-b to fall below the limit, where it is no longer considered as useful and, thus, not used.

11.5.4.2 Validity Time Attack

A similar attack – with respect to the interval time attack – uses the validity time included in HELLO and TCs. The validity time defines how long the information contained in the message should be considered as valid. After this time, the receiving router must consider the message content to no longer be valid (unless repeated in a later message) [36].

A malicious router, X, can spoof the identity of a and send a HELLO using a low validity time (*e.g.* 1 ms). b, receiving this, will discard the information upon expiration of that interval, *i.e.* a link a-b will be “torn down” by X.

11.5.5 Indirect Jamming

Indirect Jamming is an attack in which a malicious router is, by its actions, causing legitimate routers to generate inordinate amounts of control traffic, thereby increasing both channel occupation and the overhead incurred in each router for processing this control traffic. This control traffic will be originated from legitimate routers, thus to the wider network, the malicious device may remain undetected.

The general mechanism whereby a malicious device can cause indirect jamming is for it to participate in the protocol by generating plausible control traffic, and to tune this control traffic to in turn trigger receiving routers to generate additional traffic. For OLSRv2, such an indirect attack can be directed at, respectively, the Neighborhood Discovery mechanism and the Link State Advertisement mechanism.

11.5.5.1 Indirect Jamming: Neighborhood Discovery

An indirect jamming attack on the Neighborhood Discovery process is illustrated in figure 11.11.

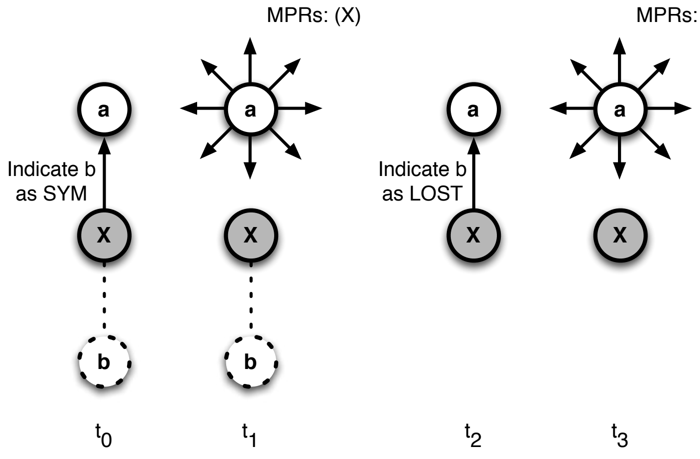


Figure 11.11: Indirect Jamming in Neighborhood Discovery: the malicious X spoofs a link to b, flipping between link status "SYM" and "LOST".

A malicious router, X, advertises in a HELLO that it has a link to b, with status SYM (t_0). This will cause a, upon receiving this HELLO, to consider b as a 2-hop neighbor, and recalculate its MPR set – selecting X as MPR. This MPR selection is signaled by a in a subsequent HELLO (t_1). Upon receipt of this HELLO from a, X advertises in a HELLO that the link to b is LOST (t_2). This will cause a, upon receiving this HELLO, to no longer consider b as a 2-hop neighbor, and recalculate its MPR set accordingly, *i.e.* to no longer contain X. This new MPR set in a is signaled in a subsequent HELLO (t_3). Upon X having received this HELLO from a, it may repeat the cycle, alternating advertising the link X-b as LOST and SYM.

In order to maximize the impact of the disruption caused by this attack, X should ensure that the router, to which it alternatively advertises a link as SYM or LOST, is not otherwise present in the 2-hop neighborhood – for example by advertising a router not otherwise present in the network. That way, all neighbors receiving a HELLO from X will select X as MPR. A way of accomplishing this is to have X learn all identities in the network by overhearing all TCs – and selecting (spoofing) an identity not already present. X will indicate its willingness to be non-zero (thus, accepting being selected as MPR) and participate in the Neighborhood Discovery procedure – and may ignore all other protocol operations, while still remaining effective as an attacker.

An easier version of this attack is to have X simply be present in the network, and participate in the Neighborhood Discovery procedure. Without spoofing a link to another router, X alternates its willingness as advertised in successive HELLO transmissions between zero (will never be selected as MPR) and the maximum value (will always be selected as MPR). The impact of this version of the attack is as above: MPR set recalculation and advertisement by neighbors of the X.

The basic Neighborhood Discovery process of OLSRv2 employs periodic message emissions, and by this attack it can be ensured that for each message exchange between X and a, the MPR set in a is recalculated. As calculation of an optimal MPR set is known to be NP-hard [160], this alone may cause internal resource exhaustion in a.

If the routers in the network have "triggered HELLOs" enabled, and that such are triggered by MPR set updates (as suggested in section 9 in [39]) this attack may also cause an increased HELLO frequency. A minimum message interval (typically much smaller than the regular periodic message interval) is imposed, to rate-limit worst-case message emissions. This attack can cause the HELLO interval to, permanently, become equal to the minimum message interval. [39] proposes that default that the minimum HELLO interval be $\frac{1}{4}\cdot$ HELLO interval.

Indirect Jamming of the Neighborhood Discovery process by a malicious router can thus have two effects: to cause increased frequency of HELLO generation and transmission by neighbors of the malicious router, *i.e.* up to two hops away from the malicious router, and to cause additional MPR set calculation in the routers which are neighbors of a malicious router.

11.5.5.2 Indirect Jamming: Link State Advertisement

The most efficient Indirect Jamming attack in OLSRv2 is to target control traffic, destined for network-wide diffusion. This is illustrated in figure 11.12.

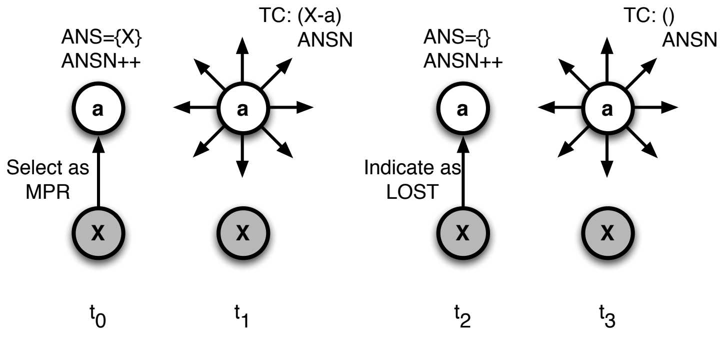


Figure 11.12: Indirect Jamming in Link State Advertisement: the malicious X flips between link status "MPR" and "LOST".

The malicious X selects a as MPR (t_0) in a HELLO. This causes X to appear as MPR selector for a and, consequently, a sets X to be advertised in its "Neighbor Set" and increments the associated "Advertised Neighbor Sequence Number" (ANSN). a must, then, advertise the link between itself and X in subsequent outgoing TCs (t_1), also including the ANSN in such TCs. Upon X having received this TC, it declares the link between itself and a as no longer valid (t_2) in a HELLO (indicating the link to a as LOST). Since only symmetric links are advertised by OLSRv2 routers, a will upon receipt hereof remove X from the set of advertised neighbors and increment the ANSN. a will then in subsequent TCs advertise the remaining set of advertised neighbors (*i.e.* with X removed) and the corresponding ANSN (t_3). Upon X having received this information in another TC from a, it may repeat this cycle, alternating advertising the link a-X as "LOST" and as "MPR".

Routers receiving a TC will parse and process this message, specifically updating their topology map as a consequence of successful receipt. If the ANSN between two successive TCs from the same router has incremented, then the topology has changed and routing tables are to be recalculated. This is a potentially computationally costly operation [47].

A malicious router may choose to conduct this attack against all its neighbors, thus attaining maximum disruptive impact on the network with relatively little overhead of its own: other than participating in the Neighborhood Discovery procedure, the malicious router will monitor TCs generated by its neighbors and alternate the advertised status for each such neighbor, between "MPR" and "LOST". The malicious router will indicate its willingness to be zero (thus, avoid being selected as MPR) and may ignore all other protocol operations, while still remaining effective as an attacker.

The basic operation of OLSRv2 employs periodic message emissions, and by this attack it can be ensured that each such periodic message will entail routing table recalculation in all routers in the network.

If the routers in the network have "triggered TCs" enabled, this attack may also cause an increased TC frequency. Triggered TCs are intended to allow a (stable) network to have relatively low TC emission frequencies, yet still allow link breakage or link emergence to be advertised through the network rapidly. A minimum message interval (typically much smaller than the regular periodic message interval) is imposed, to rate-limit worst-case message emissions. This attack can cause the TC interval to, permanently, become equal to the minimum message interval. [39] proposes as default that the minimum TC interval be $\frac{1}{4}\cdot$ TC interval.

Indirect Jamming by a malicious router can thus have two effects: it may cause increased frequency of TC generation and transmission, and it will cause additional routing table recalculation in all routers in the network.

11.6 Inconsistent Topology

Inconsistent topology maps can occur by a malicious router employing either of identity spoofing or link spoofing for conducting an attack against an OLSRv2 network.

11.6.1 Identity Spoofing

Identity spoofing can be employed by a malicious router via the Neighborhood Discovery process and via the Link State Advertisement process; either of which causing inconsistent topology maps in routers in the network.

11.6.1.1 Inconsistent Topology Maps due to Neighborhood Discovery

Considering the network in figure 11.13, two routers in far ends of the network both present themselves under the same identity – as x. The routers adjacent to these two routers (a and w in figure 11.13) both perceive x as a direct neighbor, which will be reflected in the neighbor tables and routing tables of these two routers. Thus, the first consequence is, that traffic destined for x from a and w, respectively, will be delivered to different routers. As the Neighborhood Discovery procedure provides topological information up to two hops away, this is also true for traffic destined for x from b and v, respectively.

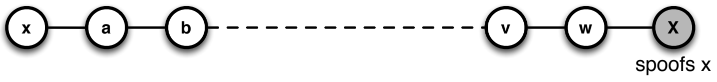


Figure 11.13: Identity Spoofing: a router (gray circle) assumes the identity of router x, located far away in the network.

Assuming unit-cost links, the distance to x from a and w, as produced by the Neighborhood Discovery procedure, is 1 hop. The distance to x from b and v, as produced by the Neighborhood Discovery procedure, is 2 hops. As these distances are shorter than (or equal to) the path lengths obtained via the Link State Advertisement procedure, they will therefore be preferred by a, b, v and w, over those acquired via the Link State Advertisement procedure for when calculating routing tables. Thus, if the gray router X in figure 11.13 is the one spoofing the identity of the white router x, then any traffic from or transiting through w and destined for x will be delivered to the gray router X instead of to the white x.

This has as impact that a router spoofing the identity of another router, and by simply participating in the Neighborhood Discovery procedure, will be able to alter the topology maps in routers up to 2 hops away, and thereby (i) attract the traffic from or transiting through routers up to two hops away, which is otherwise destined for the router whose identity is being spoofed; and (ii) prevent traffic from or transiting through routers up to two hops away, which is otherwise destined for the router whose identity is being spoofed, from reaching the intended destination.

Strategic placement of a malicious router spoofing the identity of another router (or other routers) in the network, and simply participating only in the Neighborhood Discovery process, can thereby efficiently disrupt network connectivity. First, overhearing TCs will allow the router to "learn" sufficient information describing the network topology to develop an attack strategy which has maximum disruptive impact. Second, by participating only in the Neighborhood Discovery procedure (*i.e.* by advertising its willingness as zero, and by not selecting MPRs), and by carefully selecting the identities to spoof, the malicious router can remain difficult to detect.

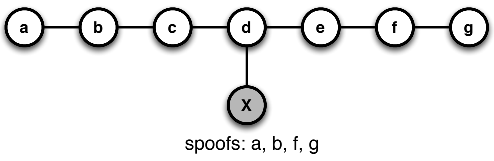


Figure 11.14: Identity Spoofing: maximizing disruptive impact while minimizing risk of detection.

Consider the example in figure 11.14: X overhears and learns the network topology. In order to minimize the risk of detection, it elects to not select any MPRs (thereby no Link State Advertisements are sent, advertising its presence in the network) and advertises its willingness as zero (thereby it is not selected as MPR and thus not required to send Link State Advertisements). X also elects to spoof the identity of a, b, f and g only. As X does not participate in the Link State Advertisement process, its presence is known only to c, d and e, *i.e.* the routers whose identity it spoofs will not receive control messages allowing them to detect that these identities are *also* advertised elsewhere in the network. Traffic transiting d, from either side, to destinations a, b, f and g will, rather than being forwarded to the intended destination, be delivered to X. Traffic transiting c and with b as destination will be delivered to the intended router b. Traffic transiting c and with a as destination may be delivered to the intended router a via b or to X via d – as the paths will be of equal length.

In figure 11.14, c is the only router which will receive control traffic indicating two topological locations of the identities a, b. However, especially in a wireless environment, this is not in and by itself unusual: a valid link might indeed exist between a and d as well as between b and d, *e.g.* through another wireless channel. Thus, the topology as perceived by c and e does not appear "improbable".

If the network grows to the left of a or to the right or g, all X has to do to continue disrupting the network is to "learn" the identities of the routers beyond a and g and also spoof the identities of these. In general, for maximum disruptive impact and minimum visibility, the malicious router would select to spoof the identities of all routers which are topologically 3 hops or more away from itself.

Identity spoofing by a malicious router, strictly participating only in the Neighborhood Discovery process, thus, creates a situation wherein two or more routers have substantially inconsistent topology maps: traffic for an identified destination is, depending on where in the network it appears, delivered to different routers.

11.6.1.2 Inconsistent Topology Maps due to Link State Advertisements

An inconsistent topology map may also occur when the malicious router takes part in the Link State Advertisement (LSA) procedure, by selecting a neighbor as MPR, which in turn advertises the spoofed identities of the malicious router. This attack will alter the topology maps all routers of the network.

In figure 11.15, X spoofs the address of a. If X selects f as MPR, all routers in the network

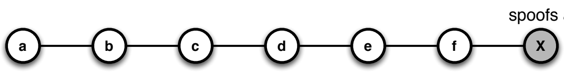


Figure 11.15: Identity Spoofing: malicious router *X* spoofs the identity of *a*, leading to a wrongly perceived topology.

will be informed about the link *f*-*a* by the TCs originating from *f*. Assuming that (the real) *a* selects *b* as MPR, the link *b*-*a* will also be advertised in the network, resulting in a perceived topology as depicted in figure 11.16.

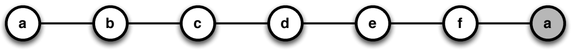
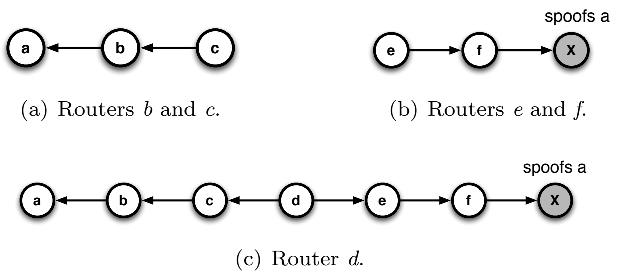


Figure 11.16: Identity Spoofing: the gray malicious router spoofs the identity of router *a*.

When calculating paths, *b* and *c* will calculate paths to *a* via *b*, as illustrated in figure 11.17(a); for these routers, the shortest path to *a* is via *b*. *e* and *f* will calculate paths to *a* via *f*, as illustrated in figure 11.17(b); for these routers, the shortest path to *a* is via the malicious router *X*, and these are thus disconnected from the real *a*. *d* will have a choice: the path calculated to *a* via *b* is of the same length as the path via the malicious router *X*, as illustrated in figure 11.17(c).



(a) Routers *b* and *c*. (b) Routers *e* and *f*.

(c) Router *d*.

Figure 11.17: Routing paths towards *a*, as calculated by the different routers in the network in presence of a malicious router *X*, spoofing the address of *a*.

In general, the following observations can be made:

- The network will be split in two, with those routers closer to *b* than to *X* reaching *a*, whereas those routers closer to *X* than to *b* will be unable to reach *a*.
- Routers beyond *b*, *i.e.* routers beyond one hop away from *a* will be unable to detect this identity spoofing.

The identity spoofing attack via the Link State Advertisement procedure has a higher impact than the attack described in section 11.6.1.1, since it alters the topology maps of all routers in the network, and not only in the 2-hop neighborhood. However, the attack is easier to detect by other routers in the network. Since the malicious router is advertised in the whole network, routers whose identities are spoofed by the malicious router can detect the attack. For example, when *a* receives a TC from *f* advertising the link *f*-*a*, it can deduce that some entity is injecting incorrect Link State information as it does not have *f* as one of its direct neighbors.

As the malicious router *X* does not itself send the TCs, but rather, by virtue of MPR selection, ensures that the addresses it spoofs are advertised in TCs from its MPR selector *f*, the attack may be difficult to counter: simply ignoring TCs that originate from *f* may also suppress the link state information for other, legitimate, MPR selectors of *f*.

Identity spoofing by a malicious router, participating in the Link State Advertisement process by selecting MPRs only, thus, creates a situation wherein two or more routers have substantially inconsistent topology maps: traffic for an identified destination is, depending on where in the network it appears, delivered to different routers.

11.6.2 Link Spoofing

Link Spoofing is a situation in which a router advertises non-existing links to other routers (possibly not present in the network). Essentially, TCs and HELLOs both advertise links to direct neighbor routers, with the difference being the scope of the advertisement. Thus, link spoofing consists of a malicious router, reporting that it has as neighbors routers which are, either, not present in the network, or which are effectively not neighbors of the malicious router.

It can be noted that a situation similar to Link Spoofing may occur temporarily in an OLSR or OLSRv2 network without malicious routers: if *a* was, but is no more, a neighbor of *b*, then *a* may still be advertising a link to *b* for the duration of the time it takes for the the Neighborhood Discovery process to determine this changed neighborhood.

In the context of this chapter, Link Spoofing refers to a persistent situation where a malicious router intentionally advertises links to other routers, for which it is not a direct neighbor.

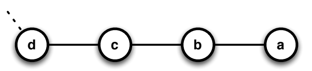
11.6.2.1 Inconsistent Topology Maps due to Neighborhood Discovery

Returning to figure 11.4(b), MPR selection serves to identify which routers are to advertise which links in the network as part of the Link State Advertisement process. OLSRv2 stipulates that a router must, as a minimum, advertise links between itself and its MPR selectors, *i.e.* links between itself and the routers which have selected it as MPR. A router is not required to advertise other links. Thus, in the example network in figure 11.4(b) with *a* selecting the malicious router *X* as its sole MPR, only *X* is expected to advertise links to *a*. *s* selects *a* as its MPR, thus *a* is expected to advertise the link *a*-*S*. *s*, then, expects *a* to have selected suitable MPRs for the MPR flooding process to succeed in network-wide diffusion of the advertisement of the link *a*-*s*.

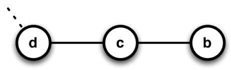
The topology maps acquired by the various other routers in this example are:

- **Routers *a* and *b*** will, due to the Neighborhood Discovery process providing topological information up to 2 hops away, acquire an accurate Topology Map. For *a* this is exactly corresponding to the network in figure 11.4(b). For *b* this may or may not contain the dotted routers *c* and *w*, depending on whether *X* generates Link State Advertisements (see section 11.6.2.2).

- **Router *c*** will perceive a topology map as illustrated in figure 11.18(a): the link state advertisements from *a* are not forwarded by *b*, hence the existence of *s* and the link *a*-*s* is not known beyond *b*; the same is true for a link state advertisement from *X*, should it participate in the link state advertisement process. The link *b*-*a*, and the existence of *a* is known to *b* only through the Neighborhood Discovery process.
- **Routers *d* and beyond** will receive a Topology Map as illustrated in figure 11.18(b).
- **Router *s*** will acquire an accurate Topology Map corresponding to the network in figure 11.4(b). This may or may not contain the dotted routers *c* and *w*, depending on if router *X* generates Link State Advertisements (see section 11.6.2.2).



(a) Inconsistent Topology Map in *c*.



(b) Inconsistent Topology Map in *d* and beyond.

Figure 11.18: Perceived Topology Maps with *X* performing Link Spoofing in the Neighborhood Discovery Process

In order to maximize the impact of the disruption of Link Spoofing in the Neighborhood Discovery process, the malicious router may simultaneously "spoof" links to multiple routers: by overhearing control traffic "for a while", *X* may attempt to learn the identities of 2-hop neighbors of *a* and spoof these – and, in addition, assume at least one additional identity (possibly not otherwise present in the network). A way of achieving this is to simply have the malicious *X* overhear all TCs, and spoof links to all identities of all routers in the network, plus one identity not otherwise present in the network. As the set of links spoofed by *X* is thus a superset of the 2-hop links as seen from *a*, *a* will select *X* as its sole MPR.

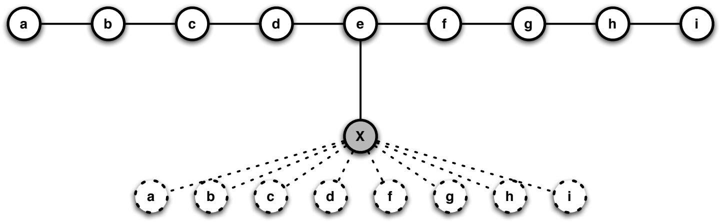


Figure 11.19: Link Spoofing: Malicious router *X* spoofs links to all routers in the network except router *e*. By only participating in the Neighborhood Discovery and not the LSA process, it is more difficult to detect the spoofing, while in the same time disturbing the topology of the whole network.

Symmetric to figure 11.14, figure 11.19 illustrates a network with *X* is positioned in the middle. If *X* advertises links to *a*, *b*, *c*, *d*, *f*, *g*, *h* and *i* in the Neighborhood Discovery process, these identities as spoofed by *X* are visible only to *e* as 2-hop neighbors. *e* may detect that no link to *X* is advertised by its own 1-hop neighbor routers *d* and *f*. Thus, to avoid such detection by *e*, *X* should avoid spoofing links to routers advertised as 1-hop neighbors by *e*, *i.e.* advertise in its HELLOs only spoofed links to *a*, *b*, *c*, *g*, *h* and *i*, as illustrated in figure 11.20.

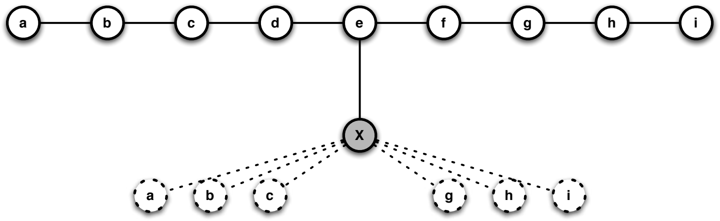


Figure 11.20: Link Spoofing: *X* remains more difficult to be detected when it does not advertise spoofed links to routers at most 2 hops away.

The impact of this this attack is that:

- *X* will appear as the most attractive candidate MPR for *e*, by virtue of spoofing links to all other 2-hop neighbors of *e*. Thus, absent *d* or *f* indicating a maximum willingness, *X* will be selected as the sole MPR of *e*.
- No routers, other than *X*, will be requested to send Link State Advertisements, advertising links to *a*.
- No routers, other than *X*, will be requested to forward flooded traffic originating in or transiting through *a*.

11.6.2.2 Inconsistent Topology Maps due to Link State Advertisements

Figure 11.21 illustrates a network, in which the malicious router *X* spoofs links to the existing router *a* by participating in the Link State Advertisement process and including this non-existing link in its advertisements.

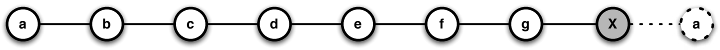


Figure 11.21: Link Spoofing: The malicious router *X* advertises a spoofed link to *a* in its TCs, thus all routers will record both of the links *X*-*a* and *b*-*a*.

As TCs are flooded through the network, all routers will receive and record information describing a link *X*-*a* in this link state information. If *a* has selected router *b* as MPR, *a* will likewise flood this link state information through the network, thus all routers will receive and record information describing a link *b*-*a*.

When calculating routing paths, *b*, *c* and *d* will calculate paths to *a* via *b*, as illustrated in figure 11.22(a); for these routers, the shortest path to *a* is via *b*. *f* and *g* will calculate paths to *a* via *X*, as illustrated in figure 11.22(b); for these routers, the shortest path to *a* is via *X*, and

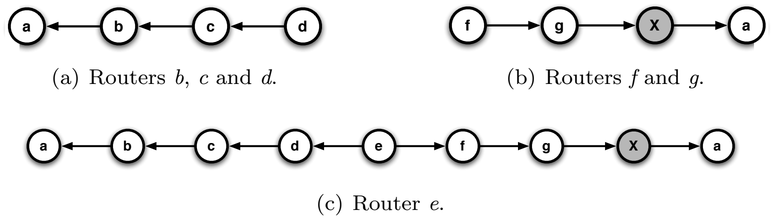


Figure 11.22: Routing paths towards router *a*, as calculated by the different routers in the network in presence of a malicious router *X*, spoofing a link to router *a*.

these are thus disconnected from the real router *a*. *e* will have a choice: the path calculated to *a* via *b* is of the same length as the path via *X*, as illustrated in figure 11.22(c).

In general, the following observations can be made:

- The network will be separated in two, with those routers closer to *b* than to *X* reaching *a*, whereas those routers closer to *X* than to *b* unable to reach *a*.
- Routers beyond *b*, *i.e.* routers beyond one hop away from *a* will be unable to detect this link spoofing.

Returning to figure 11.19, if *X* advertises spoofed links to *a*, *b*, *c*, *d*, *f*, *g*, *h* and *i* in Link State Advertisements, the risk of detection by *e* is identical to if these were advertised in the Neighborhood Discovery process: *e* may detect that *X* is advertising links to *d* and *f*, while *X* is not recorded as a 2-hop neighbor via neither *d* nor *f*.

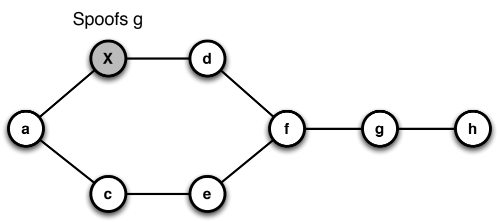
Suppressing links to *d* and *f* from being advertised by the *X* would prevent *e* from detecting that *X* is malicious. However, upon receiving a Link State Advertisement, a router is able to detect if it itself is being spoofed – the advertising router is not a neighbor of the router being spoofed. Furthermore, for the reasons elaborated above, routers up to one hop away from the spoofed destination may detect the spoofing. In the case of figure 11.20, *d* would be able to detect spoofing of links to *c* (as would *c* be able to detect spoofed links to *b* etc.) – possibly leading to a significant fraction of routers being able to detect that *X* is conducting a disruptive attack and, therefore, engaging appropriate countermeasures. *e* would, in this case, be the only router unable to detect the spoofing. While this might suffice to disrupt the network, it is no different from the identity spoofing attack illustrated in figure 11.14, which carries less risk of detection of the malicious router.

The impact of this attack is similar to that presented in section 11.6.1.2, however, is easier to detect as the malicious router is generating control traffic reaching the entire network.

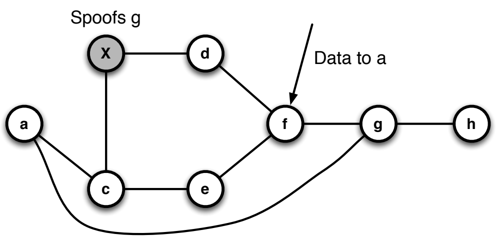
11.6.3 Creating Loops

Consider the example in figure 11.23(a). The malicious router, *X*, spoofs the identity of *g*, and participates (with this spoofed identity) in both the Neighborhood Discovery process and the Link State Advertisement process. In order to cover all its 2-hop neighbors, *a* must select both

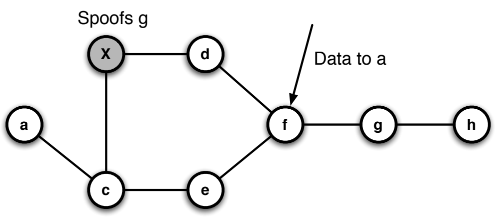
X and *c* as MPRs. Hence, the link *c*-*a* is advertised by *c*, and the link *g*-*a* is advertised by the malicious router *X*.



(a) Real network topology, with malicious router *X* spoofing the identity of *g*.



(b) Topology Map in *f*.



(c) Topology Map in *g*.

Figure 11.23: Perceived Topology Maps with malicious router *X* performing Identity Spoofing in the Link State Advertisement process

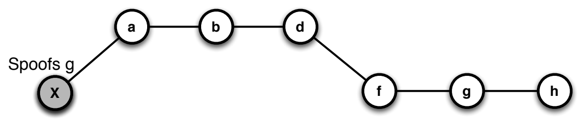
The topology perceived by *f* is as indicated in figure 11.23(b): paths to the destination *a* exist via *g* (2 hops) or via *e* (3 hops). The topology perceived by *g* is as indicated in figure 11.23(c): as *g* does not process TCs originating from itself, the only path recognized by *g* towards *a* is via *f*. Therefore, if a data packet destined for *a* arrives at *f*, it will be forwarded through *g*. *g* will forward the data packet through *f*, thereby creating a loop in the network.

Consider the example in figure 11.24(a). The malicious router, *X*, spoofs the identity of *g*, and selects *a* as MPR. Hence, the link *a*-*g* is advertised by *a* – *a* is "tricked" into advertising a non-existing link.

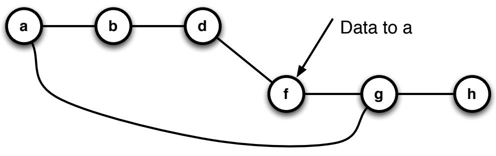
The topology perceived by *f* is as indicated in figure 11.24(b): paths to the destination *a* exists is via *g* (2 hops) or via *f* (3 hops). The topology perceived by *g* is as indicated in figure 11.24(c): as *g* does not process TCs originating from itself, the only path recognized by *g* towards *a* is via *f*. Therefore, if a data packet destined for *a* arrives at *f*, it will be forwarded through *g*. *g* will forward the data packet through *f*, thereby creating a loop in the network.

11.7 Replay Attacks

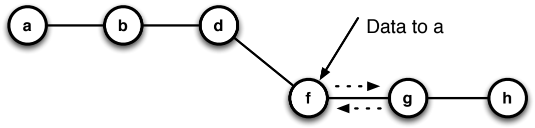
A commonly considered "attack" type is for a malicious router to record control traffic from legitimate routers, and "replay" this – possibly somewhere else in the network, and possibly at some later point in time. While such indeed is possible, it should not be considered as a class of attacks on OLSRv2 in and by itself:



(a) Real network topology, with malicious router *X* spoofing the identity of *g*.



(b) Topology Map in router *f*.



(c) Topology Map in router *g*.

Figure 11.24: Perceived Topology Maps with malicious router *X* performing Identity Spoofing in the Neighborhood Discovery process

- The malicious router replaying messages is performing a combination of *identity-spoofing*, spoofing the identity of the router from which it recorded the messages, and *link-spoofing*, spoofing links to the (original) neighbors of that router. Thus, the impact of such a "replay attack" is no different from the impact described for identity-spoofing and link-spoofing.

11.8 Inherent OLSRv2 Resilience

While OLSRv2 does not specifically include security features (such as encryption), the protocol and its algorithms present some inherent resilience against part of the attacks described in this chapter. In particular, it provides the following resilience:

- *Sequence numbers:* OLSRv2 employs message sequence numbers, specific per router identity and message type. Routers keep an "information freshness" number (ANSN), incremented each time the content of a Link State Advertisement from a router changes. This allows rejecting "old" information and duplicate messages, and provides some protection against "message replay". This, however, also presents an attack vector (section 11.5.3).

- *Ignoring uni-directional links:* The Neighborhood Discovery process detects and admits only bi-directional links for use in MPR selection and Link State Advertisement. Jamming attacks (section 11.4.2) may affect only *reception* of control traffic, however OLSRv2 will correctly recognize, and ignore, such a link as not bi-directional.

- *Message interval bounds:* The frequency of control messages, with minimum intervals imposed for HELLO and TCs. This may limit the impact from an indirect jamming attack (section 11.5.5).

- *Additional reasons for rejecting control messages:* The OLSRv2 specification includes a list of reasons, for which an incoming control message should be rejected as malformed – and allows that a protocol extension may recognize additional reasons for OLSRv2 to consider a message malformed. This allows – together with the flexible message format [35] – addition of security mechanisms, such as digital signatures, while remaining compliant with the OLSRv2 standard specification.

11.9 Conclusion

This chapter has presented a detailed analysis of security threats to OLSRv2, by taking an abstract look at the algorithms and message exchanges that constitute the protocol, and for each protocol element identifying the possible vulnerabilities and how these can be exploited. In particular, as link-state protocol, OLSRv2 assumes that (i) each router can acquire and maintain a topology map, accurately reflecting the effective network topology; and (ii) that the network converges, *i.e.* that all routers in the network will have sufficiently identical (consistent) topology maps. An OLSRv2 network can be effectively disrupted by breaking either of these assumptions, specifically (a) routers may be prevented from acquiring a topology map of the network; (b) routers may acquire a topology map, which does not reflect the effective network topology; and (c) two or more routers may acquire substantially inconsistent topology maps.

The disruptive attacks to OLSRv2, presented in this chapter, are classified in either of these categories. For each, it is demonstrated, whether OLSRv2 has an inherent protection against the attack. This vulnerability analysis sets the stage for specifying a security mechanism in chapter 12, allowing to protect the network integrity in network deployments outside of research testbeds, where security attacks are likely.

The contributions of this chapter have been published in [51, 52].

Chapter 12

Router and Link Admittance Control in OLSRv2

Chapter 11 has analyzed OLSRv2 in terms of security vulnerabilities, and proposed a "cookbook" of attacks against the stability of the routing protocol. Many of these attacks are possible only because OLSRv2 routers per default "trust" each other. Network integrity in routed networks is largely preserved by physically controlling access to the communications channel between routers: know your peers, trust your peers — and be able to disconnect your peers if they are not worthy of the trust, *e.g.* if the topology they present does not match expectations. This implies that routing integrity is protected by admitting only trusted peers, assuming that these, once admitted, are well behaving.

In a MANET, often operated over wireless interfaces, this is less obvious: physical access to the media between routers is not delimited by a cable, but is available to anyone within transmission range; the network topology is time-varying, either due to router mobility or due to time-varying characteristics of the channel – consequently, determining that a peer does not present an "expected topology" and subsequently "disconnecting" it is difficult. As such, MANETs do not introduce particularly new security issues for routing protocols, but rather render existing security issues easier to exploit and, therefore, require re-examining counter-measures for routing protocol resilience.

This chapter presents a security extension to OLSRv2 allowing to control access of routers to the network, and evaluates the increasing bandwidth and CPU resource consumption with increasing number of routers in the network.

12.1 Chapter Outline

Section 12.2 motivates the mechanism presented in this chapter. Section 12.3 describes a basic router admittance control mechanism for OLSRv2. Section 12.4 introduces a link admittance control mechanism for OLSRv2, allowing per-link verification without assuming transitive trust.

Section 12.5 specifies the protocol extension, notably the TLVs and their content, necessary for enabling router and link admittance control. Moreover, the section describes how the proposed extensions integrate into the OLSRv2 protocol architecture. As the protocol extensions proposed in this chapter rely on cryptographic signatures, section 12.6 briefly discusses the applicability of shared and public key cryptographic systems for this purpose, and section 12.7 discusses the use of timestamps in the protocol extensions proposed. Section 12.8 studies the performance of the proposed security mechanisms, with particular emphasis on (i) control traffic overhead incurred, and (ii) additional in-router resource requirements, as a consequence of these security mechanisms. This chapter is concluded in section 12.9.

12.2 Problem Statement

Consider figure 12.1, in which router *a* selects *b* as MPR in order to cover *c*. Router *b*, therefore, advertises the link *b*-*a* in TCs, throughout the network. If a malicious router, *X* (gray circle) is a neighbor of *a* and spoofs the identity of *c* (more generally, of all neighbors of b), then *a* will not select *b* as MPR. This has as consequences that (i) *b* will not advertise *b*-*a*; and (ii) the MPR flooding process is disrupted: TCs transiting through *a* will not be relayed by *b* to reach the right-hand side of the network. This is an illustration of the effect of *identity spoofing.*

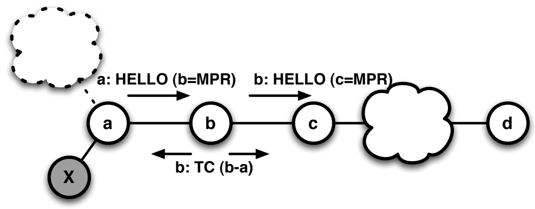


Figure 12.1: OLSRv2 overview

A possible countermeasure to such an identity spoofing attack is for a protocol extension to admit only control messages originating from routers, whose identity can be verified to not be spoofed, for processing by OLSRv2 – *router admittance control.*

Router admittance control assumes a *transitive trust relationship* between routers: *d* receiving a TC from *b* declaring a link *b*-*a*, and which *d* (by way of a router admittance control protocol extension) is able to verify was indeed sent from *b*, will have to trust that *b* is correctly behaving (*i.e.*, has not been compromised) and that *b* has properly verified the identity of *a* (the "other end of the link", advertised in the TCs received from *b*) as well as the properties associated herewith. Router admittance control does not permit the recipient of a TC to verify that the content of the TC is valid.

Still in figure 12.1, should *b* be malicious or compromised, but still in possession of credentials to generate TCs which pass verification by a router admittance protocol extension, it might in its TCs also advertise a fictitious link *b*-*d*. *c* would receive this and, thus, transit traffic destined for *d* via *b*, rather than through "to the right" to where *d* is located. This is an illustration of the effect of *link spoofing*.

A possible countermeasure to such a link spoofing attack is for a protocol extension to admit only links, where it can be verified by the recipient that both ends have *"signed off"* for the existence of that link – *link admittance control*.

12.3 Router Admittance Control

Router admittance control in OLSRv2 is enabled in [37, 39] by allowing a protocol extension to, upon receipt of a control message and prior to the usual processing hereof, determine if the message originates from a router using a "spoofed identity". Thus, each router must be able to include sufficient credentials in each control message to allow a recipient to make such a determination.

To this end, this chapter assumes that (i) each router identity (IP address) is also associated with a cryptographic key, (ii) this key is used for generating and including a cryptographic signature in each outgoing control message, and (iii) that this signature is verified by a receiving router, prior to the control message being processed by OLSRv2. The cryptographic signature is carried in control messages by way of a TLV, specified in section 12.5.

More precisely, for router admittance control, each router will, for each outgoing control message:

- calculate `sign(ownID, TimeStamp, <msg>)`;
  where `<msg>` is the control message, including all headers, but with the mutable fields `<hop-limit>` and `<hop-count>` (if present) set to zero, and `TimeStamp` is current at the time of signature generation;
- add this signature, as well as `TimeStamp`, by way of a Signature-TLV and Timestamp-TLV (section 12.5), to the control message.

Each router will, for each incoming control message and prior to it being delivered to OLSRv2 for processing:

- verify the included Signature-TLV;
- consider the message as malformed (and, thus, prohibit its processing by OLSRv2) if either of:
  - no Signature-TLV is present in the received message;
  - the verification fails, *i.e.* the signature does not correspond to the message originator and content;
  - if clocks are synchronized and Replay Attacks are of concern, the included `TimeStamp` is "too old" (refer also to section 12.7).
- otherwise, consider the message as correctly formed according to the Router Admittance Control protocol extension.

If a message is so considered “correctly formed”, it implies that the originator of the message either is not “spoofing” its identity – or, that the originator has managed to acquire the credentials, necessary for generating a signature corresponding to a spoofed identity.

12.3.1 Resilience Evaluation

The admittance control mechanism enables routers to determine if a received control message originates from a “trusted router”, discarding information received from non-trusted routers. This leaves two remaining vulnerabilities uncovered: (i) recording control traffic from a “trusted router” for later replaying (possibly elsewhere in the network) and (ii) trusted routers misbehaving, *e.g.* by spoofing links.

For (i), a countermeasure in the form of synchronized time-stamps can be employed, included in each control message; for (ii), inclusion of signatures for both ends of an advertised link can be considered, as described in section 12.4. Synchronized time-stamps, however, out of scope of this chapter.

12.4 Link Admittance Control

In order to allow a router receiving a control message to verify “both sides” of the link, (i) both sides must be able to establish that the link exists, and (ii) information “signing off” for this must be included in the control message. The TLV format in [35] enables that that information can be associated with each address, advertised in a control message.

For each address (the *other end of the link*) advertised in a control message, the originating router includes a signature embedding its own address, the address of the peer, the timestamp of emission, and any additional attributes that the originating router has associated with that link, by way of TLV inclusion, *e.g.*, if the link is HEARD or SYM (for HELLO messages) or if an address is routable (for TC messages). The signature so included is, thus: `sign(t, ownID, peerID, own-attribute-list)`. The router also signs the message as described in section 12.3, thus notably including its own timestamp in the message as well. Additionally, the router includes the most recent signature previously received for this link from the peer, the corresponding timestamp received from the peer, as well as the attribute list for this link received from the peer (*i.e.*, the information which the peer used for calculating the signature). A router receiving such a message can, then, verify if (i) the two routers agree on the attributes associated with the link, and (ii) do so at approximately the same time[1].

Consider the example depicted in figure 12.2.

At t_0, router a sends a HELLO; it has no neighbors and thus the HELLO is empty. When b receives this HELLO, it will – as usual in OLSRv2 – advertise a as HEARD in its next HELLO, at t_1 and associate its signature `sign(`t_1`, b, a, HEARD)` to this advertisement, by way of a TLV. Any router receiving this HELLO can verify only that b claims information about the link a-b.

[1]Timestamps are included to counter replay attacks. Using timestamps requires roughly synchronized system clocks. A similar mechanism using nonces could be possible, when clocks are not assumed to be synchronized.

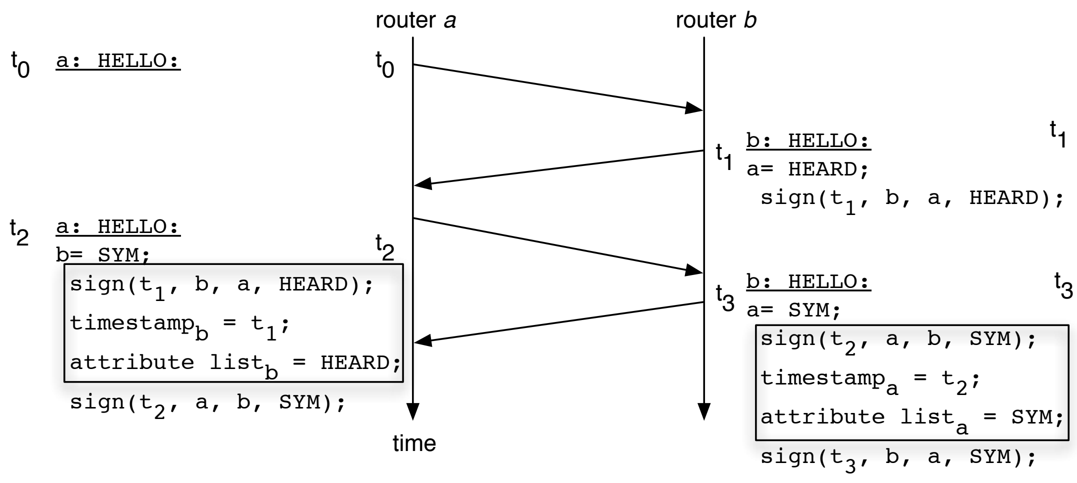


Figure 12.2: Example of link admittance control in the Neighborhood Discovery process of OLSRv2: attributes listed in “boxes” are those received from the “peer” in a previous HELLO message.

At t_2, router a will proceed in a similar fashion, advertising b as SYM, associating its own signature `sign(t₂, a, b, SYM)`. The router will also include the information shown inside the “box”: the last received signature for this link, received from b, `sign(t₁, b, a, HEARD)`, and the information necessary for a third-party to be able to verify the signature of b: the timestamp t_1, and the attribute list corresponding to this link as received from b (HEARD). Any router receiving this HELLO can by verifying the signatures observe that a and b at this point have claimed different information about a (a describes the link as SYM, b describes it as HEARD).

At t_3, router b may consider advertising a as SYM, and associating its own signature `sign(t₃, b, a, SYM)`. The router will also include the information shown inside the “box”: the last received signature for this link, received from a, `sign(t₂, a, b, SYM)` and the information necessary for a third-party to be able to verify the signature of a: the timestamp t_2, and the attribute list corresponding to this link as received from a (SYM). Any router receiving this HELLO can by verifying the signatures observe that at t_2 and t_3, respectively, both a and b have claimed claimed that the link a-b is symmetric. If t_2 and t_3 are sufficiently close, a recipient may conclude that a symmetric link exists between a-b, and that both a and b have “signed off” for this. The link can therefore be considered as “trusted”, and thus reflected in the link- and neighbor-sets of OLSRv2.

The main impact, in terms of protocol operation, is that if link admittance control is used when admitting routers to the 2-hop set, then one further HELLO message exchange is required in order for a router to be able to detect 2-hop links as “signed off” as symmetric by both ends.

12.5 Protocol Extension Specification: Router and Link Admittance Control

In the following, the router and link admittance control protocol extension, proposed in this chapter, is specified, in particular the TLV types introduced, as well as the interaction between this protocol extension and OLSRv2.

12.5.1 TLV Specification

Three TLVs are required: a timestamp TLV, a signature TLV and an attributes TLV. The timestamp TLV and the signature TLV, both, can be used as Message TLVs (*i.e.* included in the header of a control message) and as Address Block TLVs (*i.e.* associated with one or more addresses in the message body). This section specifies the content of `<value>` in the three proposed TLVs, for use in the format described in [35].

12.5.1.1 Timestamp TLV

```
<timestamp> := <time-value>
```

where: `<time-value>` contains the timestamp. A timestamp is essentially "freshness information", and may *e.g.* correspond to a UNIX-timestamp, GPS timestamp or a simple sequence number. For the performance study of section 12.8, the timestamp is a four-byte long integer, counting the seconds from the start of the simulation.

12.5.1.2 Signature TLV

```
<sign-tlv> := <hash-fkt><sign_algo><sign>
```

where: `<hash-fkt>` and `<sign_algo>` identify the choice of hash function and signature algorithm, respectively, and `<sign>` contains the digital signature, calculated thus:

```
sign = sign_algo(hash-fkt(message))
```

Verification of a message is a boolean operation, acting as a black-box for the routing protocol and returning true if the message signature verifies, false otherwise:

```
verified = verif(message, <sign-tlv>)
```

12.5.1.3 Attributes TLV

```
<attributes> := {<attribute-value>}*
```

Recalling that for each address included in a message, two signatures are ultimately included. If the message is originated by router a and contains an address of a peer b, then for the link a-b a signature generated by both a and b is included. In order for a third-party c to be able to verify also the signature from the peer b, the exact information over which the peer b calculated this signature needs to be available to c. In figure 12.2 at t_2, for example, this information is included in the box: the timestamp (t_1) and the attribute list (HEARD). This information has been received by a by way of TLVs, and the signature (`sign(`t_1`, b, a, HEARD)`) can be verified by a using the identities of both routers a and b as well as the timestamp (t_1) from the timestamp TLV and the attribute (HEARD) from the attributes TLV.

12.5.2 OLSRv2 Interaction

Due to the flexible nature of the specification of OLSRv2, the router and link admittance control extension, presented in this chapter, can be designed as an independent module. This module is invoked when an incoming control message has been successfully parsed, and before further processing by OLSRv2, as well as whenever an outgoing message is about to be sent. This simple architecture allows combination of other OLSRv2 extensions with this router and link admittance control extension, without encumbering such other extensions. The flow of incoming and outgoing messages between the different modules is depicted in figure 12.3.

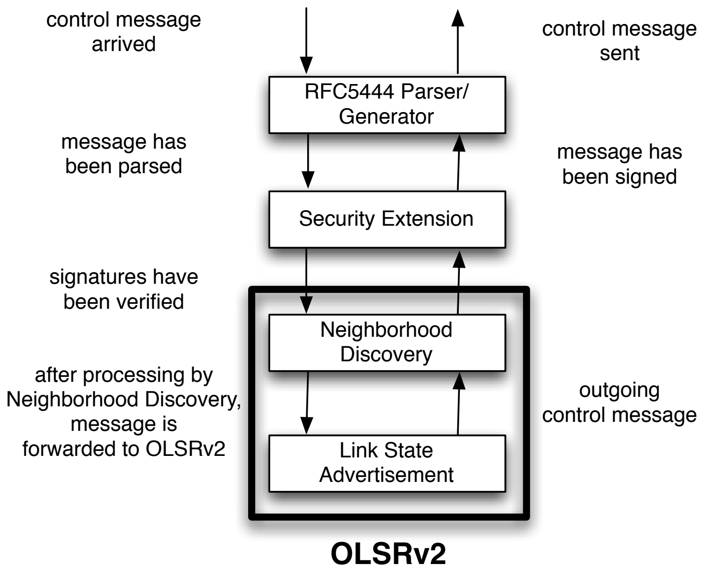


Figure 12.3: Message flow between NHDP, OLSRv2 and the security extension.

12.6 Cryptographic Keys

Using cryptographic signatures in control messages allows the recipient of a message to (i) verify the integrity upon receipt, (ii) verify if the originator is to be admitted to the network, and (iii) verify the identity of originator of the control message. For (i) and (ii), a “pre-shared secret”, such as a secret passphrase or a symmetric key, suffices: only routers possessing the secret are able to correctly sign control messages and addresses within, which allows exclusion of “all but pre-approved routers". However, (iii) requires asymmetric keys for allowing per-principal authentication. As the link admittance control extension, presented in this chapter, relies on bi-directional verification of links between routers, per-principal authentication is a requirement.

For assuring the reliability of the admittance control system, it is paramount that the cryptographic keys are only accessible by the router that they are allocated to. Furthermore, it has to be ensured that the keys cannot be derived from any information exchanged between the routers, *i.e.*, that the cryptographic algorithm is not vulnerable to attacks, other than brute-force with sufficient efforts for such a brute-force attack being appropriate for deployment requirements.

12.7 Timestamps

The router and link admittance control extension can provide protection against identity-spoofing and link-spoofing of unadmitted routers (*i.e.* routers not able to correctly sign messages), but malicious routers can still record and replay messages (refer to section 11.7 and to [51] for details on such "replay attacks"). These replay attacks can be partly avoided by introducing "freshness" information, such as timestamps or nonces, in messages. A message which is replayed some time after the recording, can be detected as being "old" (at least, when the clocks of the routers are roughly synchronized).

In the protocol extension proposed in this chapter, both whole messages and individual links are candidates for "being replayed". Consider a router X which has been compromised (*i.e.*, the attacker has access to appropriate credentials to generate correctly signed messages). If no timestamps were included in the per-link signatures, X could record such per-link signatures and include them later in its messages, spoofing links to non-existing neighbors.

12.8 Performance Study

This section presents a performance study, by way of simulations with NS2, of the link admittance control mechanism, in comparison to a simple router admittance control mechanism, both in terms of message overhead and CPU time. Simulations have been conducted with JOLSRv2 using relatively standard scenario parameters (1km^2 square, 0-300m segments of random walk at 2-8$\frac{m}{s}$, and 0-5s pause-time). The number of routers was varied between 10 and 50, and each value has been averaged over 20 simulation runs. The performance parameters studied are the extra control traffic overhead and the in-router message generation/processing overhead incurred by the mechanisms presented in this chapter. As JOLSRv2 with AgentJ permits single thread execution, without preemption, it is possible to instrument the signature generation and verification code to record the time spent on each such operation[2]. For the simulations, an Intel Core 2 CPU with 2.1 GHz and 4 GB of RAM was used.

[2]For completeness: AgentJ rewrites `System.currentTimeMillis()` such as to return the "simulator time", whereas `System.nanoTime()` is not rewritten and therefore returns the "wall clock time".

12.8.1 Overhead of Router and Link Admittance Control

In the following, the applied signature algorithms are RSA-1024 [161], DSA-1024 [162], ECDSA-160 [163], and HMAC-80 [164][3], with the numbers being the key length in bits for each respective algorithm.

For RSA, DSA and HMAC, the implementations directly provided by Java 6 have been used, whereas ECDSA is a custom implementation.

Using link admittance control, described in section 12.4, up to two signatures are included per advertised address. Thus, the message size will grow with the density of the network, as depicted in figure 12.4. Note that in this figure, as well as in the following, the overhead only considers the size of the signatures, and *not* the content of the HELLO or TC message themselves. Thus, for an unsigned message, the overhead would be 0.

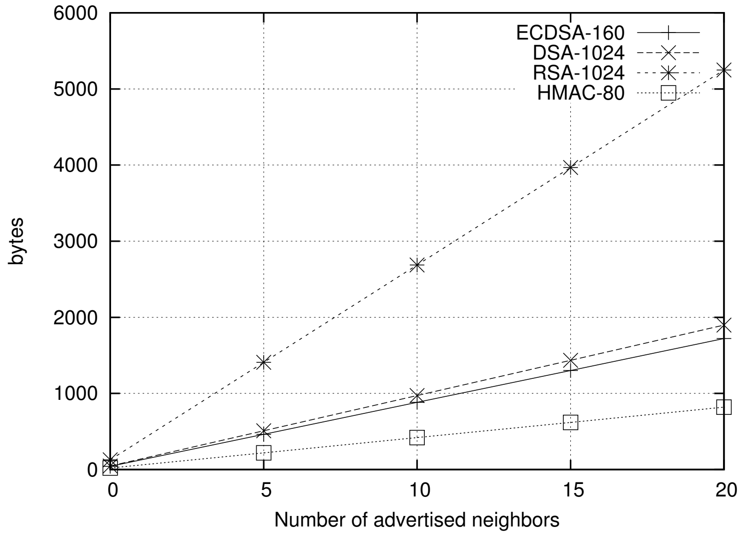


Figure 12.4: Link admittance control: Signature overhead per control message with increasing number of advertised neighbors.

While in the link admittance control mechanism, the signature overhead per message increases with the number of advertised neighbors, it remains constant when using only a router admittance control mechanism (*i.e.* one signature per message). Table 12.1 summarizes the average overhead, incurred by each of the four signature algorithms.

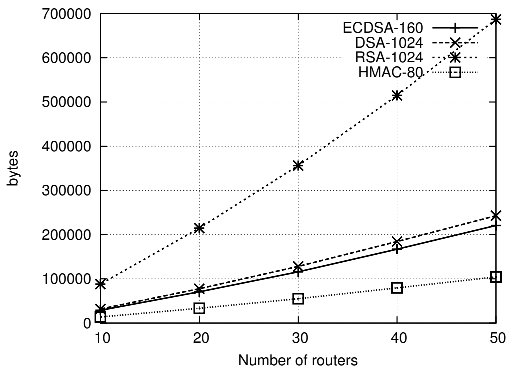
In the following, RSA, DSA, ECDSA and HMAC are considered for the router admission control comparison, but only ECDSA and RSA are included for the link admission control for reasons of clarity of the figures. The different signature mechanisms are explored in terms of (i) message overhead and (ii) CPU time for processing and generating signatures.

Figure 12.5 depicts the cumulative overhead in the network, due to inclusion of message signatures and address signatures.

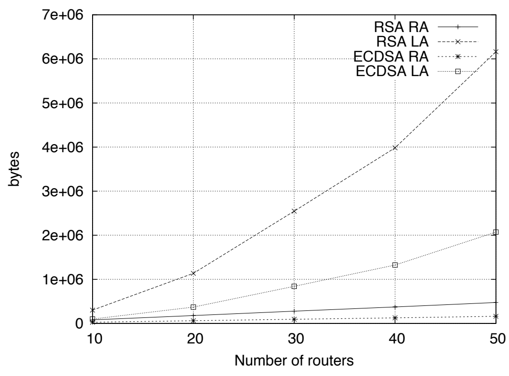
[3]Note that HMAC, strictly speaking, is not a signature algorithm, but a Message Authentication Code, see section 12.6.

Table 12.1: Simulation Results: per-message signature overhead

Signature algorithm	Overhead
ECDSA-160	42 bytes
DSA-1024	46.31 bytes
RSA-1024	128 bytes
HMAC-80	20 bytes



(a) Router admittance control only



(b) Both router and link admittance control

Figure 12.5: Total overhead incurring due to signature inclusion

For the router admittance control mechanism (denoted "RA"), the per-message overhead is constant, and the cumulative overhead is a function of the number of control messages – itself a function of simulation time and number of routers. Note that the length of the control message does not influence the length of the signature, since the signature is always calculated over an SHA1 hash of the message. With link admittance control (denoted "LA"), the total overhead grows polynomial with increased number of routers and increased density in the network, as up to two signatures are added per advertised neighbor in a control message. As RSA-1024 signatures are longer than the corresponding ECDSA-160 signatures, the total overhead with RSA grows considerably faster.

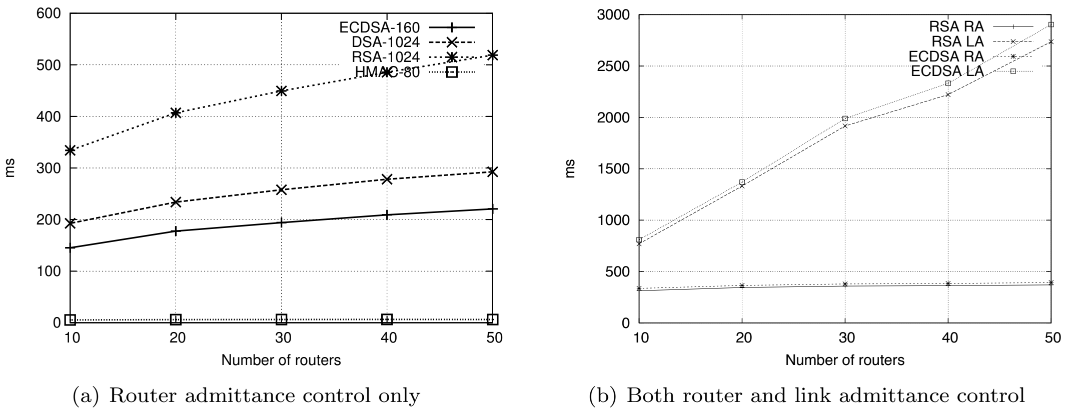
Using smaller signatures (*e.g.*, as provided by ECDSA) is, in terms of message size, particularly beneficial for link admittance control. Longer signatures (such as RSA) lead to (i) higher bandwidth consumption for control traffic, and therefore (ii) higher energy consumption[4], as well as (iii) the possibility that the IP packet gets fragmented when its size is greater than the MTU of the underlying link layer. This can be well observed in figure 12.4; assuming an MTU of 1500 bytes (*e.g.* for Ethernet), messages signed with RSA would be fragmented (with the associated risk of any one fragment being lost causing the whole message to be lost) with about five neighbors, whereas ECDSA would allow roughly three times more neighbors.

[4][165] states *"The energy cost of transmitting 1Kb a distance of 100 meters is approximately 3 Joule. By contrast, a general-purpose processor with 100MIPS/W power could efficiency execute 3 million instructions for the same amount of energy"*, indicating that shorter (but more computationally intensive) signatures for certain applications, such as energy constrained devices, may be preferential.

12.8.2 In-Router Resource Requirements

This section analyzes the CPU time for creating and parsing signatures in control messages in OLSRv2 using the router admittance and the link admittance control mechanisms respectively.

Figure 12.6 depicts the cumulative time each router spends, over the duration of 100 seconds, on generating signatures in JOLSRv2, with router admittance (denoted "RA") and link admittance (denoted "LA"), both. For router admittance control, the time each router spends in total on generating signatures is constant, since every router periodically creates HELLO and TC messages, independently of the size of the network. As expected, using link admittance control significantly increases the amount of CPU time required for generating control messages, since the number of signatures per message to be generated increases with the density of the network. ECDSA and RSA have similar time consumption for generating signatures.



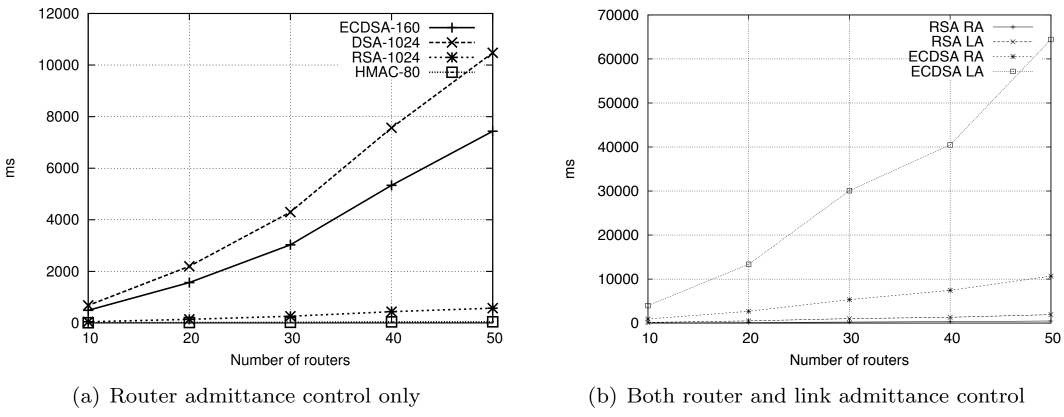
(a) Router admittance control only (b) Both router and link admittance control

Figure 12.6: Cumulative time per router spent on generating message signatures

The corresponding cumulative processing time in each router is depicted in figure 12.7. Each router generates HELLOs, which must be processed, and so its signatures verified, by its neighbors. Thus, increasing the network density increases the number of HELLOs that a given router receives and, therefore, the number of signatures to verify. Depending on the network topology and MPR selection, additional routers may also incur additional TCs, whose processing and signature verification is to be conducted by each other router in the network. Link admittance control significantly increases the amount of CPU time spent for verifying signatures for control messages. RSA signatures are fast to verify, while verifying ECDSA signatures consumes a considerable amount of CPU time. Since every signature has to be verified before it can be forwarded, the total amount of time spent in each router for verifying signatures is considerably higher than for generating messages.

12.9 Conclusion

When OLSRv2 routers use digitally signed control messages for admittance control, these routers can verify the identity of control message originators and the integrity of the messages. However, a



(a) Router admittance control only (b) Both router and link admittance control

Figure 12.7: Cumulative time per router spent on verifying message signatures

router has to trust the message originator that the advertised links in the HELLO or TC message are valid. This chapter has specified a router and link admittance control protocol extension to OLSRv2, which allows a router to verify each advertised link from incoming control messages, by signing "both ends of the link". The router and link admittance control protocol extension is generic, in that it is not tied to any specific cryptographic system. Indeed, the mechanism operates as long as the choice of cryptographic system allows for per-principal authentication and signature generation.

A performance study of this extension has been presented, quantifying the impact in terms of increased control traffic overhead and increased per-message generation and processing time, exemplified by using two relatively common cryptographic systems: RSA, for its performance in verification of signatures, and ECDSA for its short signature lengths for the same "strength" of signatures. It is argued that using shorter signatures may be advantageous when using such a router and link admittance security mechanism, since the additional overhead grows linear with the density of the network. Using longer signatures leads to (i) higher bandwidth consumption for control traffic, and therefore (ii) higher energy consumption, as well as (iii) the possibility that the IP packet gets fragmented when its size is greater than the MTU of the underlying link layer.

It is observed, however, that regardless of the choice of cryptographic system, this router and link admittance control protocol extension is no "free lunch": other than the size increase in control messages, the time required for signature generation and verification is – unsurprisingly – not negligible.

The contributions of this chapter have been published in [53, 54] and in the IETF Working Group draft [55].

Part IV

Wireless Sensor Networks

Chapter 13

RPL Protocol Overview

For deployments such as described in section 1.3.3, the IETF has created the ROLL Working Group and chartered it to work on a routing protocol for constrained sensor networks. This section outlines the basic mechanisms of this routing protocol, denoted “Routing Protocol for Low Power and Lossy Networks” (RPL) [43].

The objective of this protocol is to target WSNs which “*comprise up to thousands of routers*”, where the majority of the routers have constrained resources, where the network to a large degree is “managed” by a (single or few) central “super-routers”, and where handling mobility is not an explicit design criteria. Supported traffic patterns include multipoint-to-point, point-to-multipoint and point-to-point traffic. The emphasis among these traffic patterns is to *optimize for* multipoint-to-point traffic, to *reasonably support* point-to-multipoint traffic and to *provide basic features for* point-to-point traffic, in that order.

13.1 Chapter Outline

Section 13.2 describes the general routing structure that is used in RPL, a directed acyclic graph of routers, each associated with a “rank” (logical distance to a “root” node). Section 13.3 presents several data traffic flows for which RPL is supposed to be deployed. Section 13.4 explains certain limitations for routers to change their ranks in order to avoid count-to-infinity problems. Section 13.5 lists the minimal operational requirements for RPL. Section 13.6 describes the “trickle” timer, which is used to determine the control traffic intervals, and to minimize these intervals for optimizing the utilized bandwidth of the routing control traffic. Section 13.7 provides a critical discussion of RPL. Section 13.8 concludes the chapter.

13.2 DODAG Routing Structure

The basic construct in RPL is the DODAG – a destination oriented directed acyclic graph – rooted in a “controller” or “root”, in figure 13.1. In the converged state, each WSN router has identified a stable set of parents, on a path towards the “root” of the DODAG, as well as a *preferred parent*. Each router, which is part of a DODAG (*i.e.* has selected parents), will emit *DODAG Information Object* (DIO) messages, using link-local multicasting, indicating its respective *rank* in the DODAG (*i.e.* its position – distance according to some metric(s), in the simplest form hop-count – with respect to the root). Upon having received a (number of such) DIO messages, a router will calculate its own rank such that it is greater than the rank of each of its parents, and will itself start emitting DIO messages. Thus, the DODAG formation starts at the root, and spreads gradually to cover the whole network.

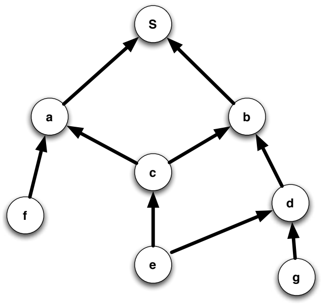


Figure 13.1: RPL basic construct: DODAGs

13.3 RPL Data Traffic Flows

RPL provides routes for three kinds of traffic flows, which are presented in this section:

13.3.1 Upward Routes

The DODAG, constructed as explained in section 13.2, is used for installing routes in the WSN routers: the "preferred parent" can serve as a default route towards the root, or the root can embed in its DIO messages the destination prefixes, also included by DIOs generated by WSN routers through the WSN, to which it can provide connectivity. Thus, RPL provides "upward routes" or "multipoint-to-point routes" from the sensors towards the root. Figure 13.2 depicts an example upward data traffic flow from a source router, denoted "s", towards the destination (*i.e.* the root), denoted "d".

13.3.2 Downward Routes

"Downward routes" are installed by having the sensors issue *Destination Advertisement Object* (DAO) messages, which propagate via "DAO" parents towards the root, and which describe which prefixes belong to, and can be reached via, which WSN router. Figure 13.3 depicts an example downward data traffic flow from the root router, denoted "s", towards the destination, denoted "d".

As an option in RPL, the successful reception of a DAO message may be acknowledged with a DAOACK message, if that is solicited (as indicated by setting the "K" flag in the DAO message).

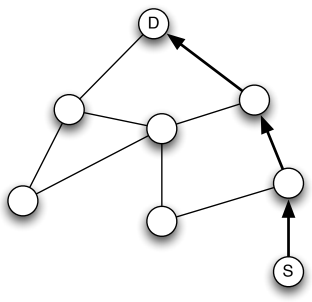


Figure 13.2: Upward routes

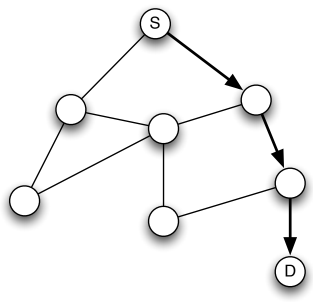


Figure 13.3: Downward routes

Two different modes for downward routes are supported: *storing* and *non-storing* mode, depending on the capacities of the routers. All routers in the WSN must operate in the same mode (besides the root, which must always be in storing mode). In the following, the two different modes are briefly presented.

13.3.2.1 Storing Mode

In *storing mode*, each router is assumed to maintain routes to all WSN routers in its sub-DODAG, *i.e.* routers that are "deeper down" in the DAG. DAOs propagate from the routers towards the root, where each intermediate router adds its reverse routing stack to the DAO message (aggregating routes where possible). When the root sends data traffic downwards, the packet is sent in a hop-by-hop way, and each router along the path looks up the next hop for the packet in its downward routing table.

Storing mode requires storage capacities on each router, for paths to possibly all other routers in the network. No source routing is required, however, such as required in non-storing mode.

13.3.2.2 Non-Storing Mode

In *non-storing mode*, only the root stores routes to all WSN routers in the network. Each WSN router unicasts DAOs to the root, which then calculates routes to all destinations by "piecing together" the information collected from DAO messages (which contain the destination prefix of the LLN router and addresses of the parents through which it is reachable). In non-storing mode, downward traffic is sent by way of source routing, *i.e.* an IPv6 Hop-by-hop header is added containing the path the packet must traverse towards the destination.

Non-storing mode does not require additional state on the routers, but increases the message overhead, as the source routes are added to data packets.

13.3.3 Sensor-to-Sensor Routes

Sensor-to-sensor routes are as default supported by having the source sensor transmit data traffic upwards to the first router that has a valid downward route towards the destination. In non-storing mode, the first router to have downward routes towards the destination is the root node, as depicted in figure 13.4. In storing mode, it is possible that some router along the path towards the root has a downward route towards the destination, in which case the shorter path via that router is used, depicted in figure 13.5.

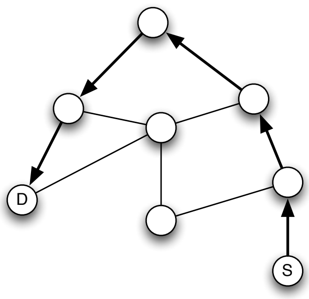


Figure 13.4: P2P data flow via the root

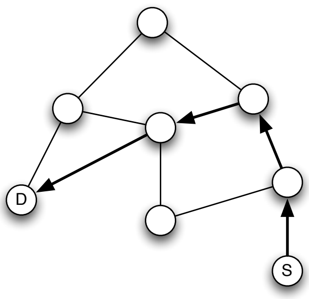


Figure 13.5: P2P data flow via the first router that has a valid downward route towards the destination (only possible in storing mode)

A more optimal sensor-to-sensor mechanism (in terms of path length) is considered in [166], a draft which is still in an early stage as of 2011.

13.4 Rank Changes in RPL

As a Distance Vector protocol, RPL contains rules restricting the ability for a router to change its rank. Specifically, a router is allowed to assume a smaller rank than previously advertised (*i.e.* to logically move closer to the root) if it discovers a parent advertising a lower rank (and it must then disregard all previous parents with higher ranks), while the ability for a router to assume a greater rank (*i.e.* to logically move farther from the root) in case all its former parents disappear, is restricted to avoid count-to-infinity problems. This is illustrated with the example in figure 13.6.

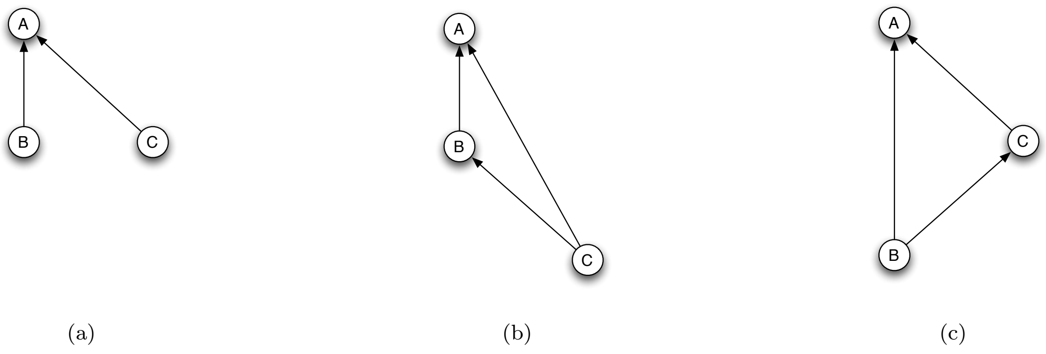


Figure 13.6: Greedy DODAG parent selection (arrows represent parent selection of a router)

Figure 13.6(a) depicts the initial situation: router B and C both have selected router A as parent (*i.e.* they have a higher rank than A). Assuming that router C increases its rank to a higher value than both A and B, as depicted in figure 13.6(b), it would select both A and C as parents. If router B is greedy and in its greediness is willing to receive and process a DIO message from router C (against the rules of RPL), it increases its rank to a higher tank than both A and C (depicted in figure 13.6(c)). Next, C would again increase its rank in order to have more parents, resulting in the situation of figure 13.6(b) again. The process will repeat, and the DODAG will oscillate between figure 13.6(b) and figure 13.6(c) until the ranks of the routers count to infinity.

In order to avoid such count-to-infinity problem, RPL binds the maximum rank increase of each rank[1]. Moreover, DIOs from routers with a higher rank are ignored.

The exact rank calculation, as well as parent and preferred parent selection, is not specified in RPL [43], but is encapsulated in a so-called "Objective Function", which is implementation and deployment specific.

13.5 RPL Operational Requirements

The minimal set of in-router state, required in a WSN router running RPL is, (i) the identifier of the DODAG root, (ii) the address and rank of the preferred parent, (iii) the configuration parameters shared by the DODAG root (notably, destination prefixes and message emission timers) and (iv) the maximum rank that the WSN router has itself advertised. For redundancy, a WSN router running RPL can maintain information describing additional parents (up to and including all its parents), which may allow rapidly changing its preferred parent (and thus its "next hop") in case the former preferred parent becomes unreachable.

13.6 Trickle Timer

RPL message generation is timer-based, with the root able to configure suitable back-off of message emission intervals using *trickle timers* [167]. Trickle timers allow for reducing the frequency of transmissions of routers, while still maintaining consistent information throughout the network. As long as a router receives information that is "consistent" with its own information, it will exponentially increase its transmission interval, until a predefined minimum frequency is reached. For RPL, "consistent" means that a DIO message from a sender with a lesser rank causes no changes to the recipient's set of parents, its preferred parent, or rank. An "inconsistency" occurs if a router detects a routing loop, or if it receives a DIO message indicating an increased DODAG version number. In case of such an event, the trickle timer will be "reset", *i.e.* the transmission interval is set to the predefined minimum.

In addition to the exponential increase of the transmission interval, if a router receives a certain predefined number of redundant (*i.e.* consistent) messages, the router may suppress its

[1]An exception is "poisoning": a router may increase its rank to "infinity", and advertise this rank in its DIOs in order to "poison" its sub-DODAG

own retransmission, in order to reduce bandwidth utilization and to save energy.

13.7 RPL Discussion

In its basic form, RPL is a fairly simple-to-understand and simple-to-implement distance-vector protocol. The DODAG formation mechanism, using DIO messages, is currently well understood, and despite the specification hereof in [43] remaining somewhat ambiguous, the implementation presented in chapter 14 was developed and tested "from scratch" within about a week.

The DODAG formation mechanism is not without potential issues, however. First, parents (and the preferred parent) are selected based on receipt of DIO messages, without verification of the ability for a WSN router to successfully communicate with the parent – *i.e.* without any bidirectionality check of links. In a wireless environment, unidirectional links are no rare occurrence, and can simply happen as illustrated in figure 13.7: the gray device, X, illustrates a source of environmental interference, preventing route b from successfully receive transmissions from a. This may, however, not prevent b from transmitting DIOs, received by a and which may contain information causing a to select b as both parent and preferred parent.

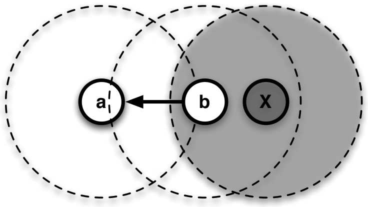


Figure 13.7: Unidirectional link due to radio interference

As b is a "useless" next-hop for a, due to the interference from X, this is a bad choice. RPL suggests using Neighbor Unreachability Detection (NUD) [86] to detect and recover from this situation, when it occurs that a tries (and fails) to actually use b for forwarding traffic. NUD is based upon observing if a data packet is making forward progress towards the destination, either by way of indicators from upper-layer protocols (such as TCP), from lower-layer protocols (such as Link Layer ACKs) or – failing these two – by unicast probing. A couple of problems can be noted regarding this approach.

First, absent all WSN routers consistently advertising their reachability through DAO messages, a protocol requiring bi-directional flows between the communicating devices, such as TCP, will be unable to operate. Even if such bi-directional flows are enabled, the source detecting, by way of an upper layer protocol, that no forward progress is possible, is of restricted use: the source can not know if it is its "preferred parent" (next hop) which is unreachable, or if it is a problem further along the path (even outside the WSN). Thus, any corrective action that the source might take (changing preferred parent, moving to a higher rank within the limits allowed, etc.) may be unable to alleviate the problem, and corrective actions may even be counter-productive (poison the sub-DODAG, for example).

Second, there is a change that the radio range of a unicast (as would be used for data delivery via the next hop towards the root) would differ from the radio range of DIOs, which are sent using link-local multicast[2].

Third, upon having been notified by NUD that the “next hop” is unreachable, a WSN router must discard the preferred parent and select another preferred parent – hoping that this time, the preferred parent is actually reachable. Also, if NUD indicates “no forward progress” based on an upper-layer protocol, there is no guarantee that the problem stems from the preferred parent being unreachable. Indeed, it may be a problem farther ahead, possibly outside the WSN, thus changing preferred parent will do nothing to alleviate the situation.

Fourth, the selection of parents and preferred parent is based on receipt of DIO messages only, and is based on the rank of the candidate parents. Absent other complementary mechanisms (which are currently not specified as part of [43]), a WSN router may receive, transiently (*e.g.* due to a fortunate environmental reflection), a DIO from another router, much closer to the root – and as a consequence change its parent set and rank to this new more attractive parent. If no stable link exist, this may cause delivery failures.

The Destination Advertisement mechanism, for providing downward routes “from the root to the sensors”, remains in a state of flux. While the basic properties of the Destination Advertisement mechanism, given a stable underlying DODAG, appear easy to understand, it does have several inconveniences: in non-storing mode, all sensor-to-sensor routes transit the root, possibly causing congestion in the wireless spectrum near the root, as well as draining energy from the intermediate routers on an unnecessarily long path. Several solutions are proposed to alleviate this, including allowing intermediate WSN routers, otherwise only forwarding DAO messages towards the root, to record routing state, and allowing these intermediate WSN routers to act as “shortcuts”. Another proposed solution is to use proper sensor-to-sensor routing protocols, derived off *e.g.* AODV [97].

Finally, the current specification of RPL does not provide support for “broadcasting” of any form. Unicast traffic to and from the root can be enabled, as previously described, however is inefficient in case the root has data to deliver to all (or a sufficiently large subset) of the WSN routers in the network.

13.8 Conclusion

The “IPv6 Routing Protocol for Low Power and Lossy Networks” (RPL) is being specified by the ROLL Working Group of the IETF, supposedly standardized in 2011. The protocol is based on a “Destination Oriented Directed Acyclic Graph” (DODAG) structure with a root router, that serves as collector of data from the sensor nodes and may provide connectivity to other IPv6 networks. RPL supports three kind of data traffic flows: it is supposed that a large fraction of traffic is directed “upwards” from the sensors towards the root, some traffic is directed “downwards” from the root to the sensors, and rarely from sensor-to-sensor.

[2]Such is the case for some implementations of IEEE 802.11b. IEEE 802.11b is, of course, not suggested as a viable radio interface for WSNs, but serves to illustrate that such asymmetric designs exist.

The DODAG is constructed by routers emitting DODAG Information Objects (DIO) from the root towards the leaves of the network, where each router calculates its rank (*i.e.* its distance to the root based on a specific metric) and selects parents from the direct neighbors with a lower rank. One of the parents is selected as preferred parent, which serves as next hop for routing data packets towards the root.

Downward routes are constructed using Destination Advertisement Object (DAO) messages. All routers in the network can either be operated in "storing" or in "non-storing" mode, where routers in the storing mode keep a downward routing table in memory, allowing a hop-by-hop routing from the root towards the destination, where each routers selects the next hop by looking up the destination in the routing table. In non-storing mode, source routing is required, *i.e.* only the root stores a downward routing table to all destinations, and adds the complete path in an IP header to data packets.

RPL only contains a basic sensor-to-sensor traffic flow mechanism, via the root or – in storing mode – via the first common ancestor of the source and the destination, that has a valid downward route towards the destination.

RPL is optimized to use little storage, and to emit as few control traffic messages as required, applying a "trickle" timer that reduces the intervals of control traffic transmission exponentially up to a minimal interval, as long as the topology remains stable. RPL does not contain any bidirectionality test of links between neighbors, possibly causing unstable topologies. The downward mechanism is complex with many changes during the specification process, and seems less well understood than upward data flows. Moreover, no "broadcasting" mechanism is currently specified in RPL, *i.e.* a mechanism to disseminate a packet from the root to all other routers.

Chapter 14

Java RPL Implementation

This chapter describes the RPL implementation in Java, denoted JRPL. The architecture is similar to JOLSRv2 (as described in chapter 7).

The implementation supports both “upward” flow and “downward” flow, in “non-storing mode” [43]. A local repair mechanism has been developed, as well as datapath loop detection. JRPL does not provide storing mode, global route repair (*i.e.* version number increase), and only supports a single RPL instance and a single DODAG. However, all data structures have been integrated to allow to easily extend JRPL to contain these features.

14.1 Chapter Outline

Section 14.2 gives a brief overview of the architecture of the implementation. Section 14.3 details the API and the behavior of the used objective function. Section 14.4 describes the upward mechanism, including DIOs and upward unicast data traffic. Section 14.5 presents the implementation of downward traffic in non-storing mode, including DAOs, DAOACKs, and unicast data traffic. Section 14.6 outlines the local repair mechanisms of JRPL, and section 14.7 presents a loop detection mechanism based on data traffic. Section 14.8 concludes the chapter.

14.2 Architecture Overview

Figure 14.1 depicts the JRPL code structure. The main class of JRPL is `net.rpl.RPLRouter`, which represents one RPL process running on a router. `net.rpl.RPLInstance` and `net.rpl.DODAG` represent the equivalent terms from the RPL specification [43]. `net.rpl.comm` contains classes related to message sending and reception, as well as a class for the trickle timer. `net.rpl.generators` contains traffic generators, that inject data traffic in the network. `net.rpl.messaging` comprises the different message types and all message options. `net.rpl.of` comprehends a basic version of the objective function (described in section 14.3), and `net.rpl.repository` consists of classes that store the state of the routing protocol. `net.rpl.ns2` is comprised of classes specific to NS2 simulations with JRPL, and `net.rpl.NUD` contains classes for Neighbor Unreachability Detection (NUD).

14.3 Objective Function

Similar to the structure of the specification [43], an abstract class containing an API for the objective function contains the methods listed in table 14.1:

Table 14.1: Implemented API of the objective function

Method	**Description**
`abstract int getMyRank();`	returns the rank of the router
`abstract DODAG selectDODAGIteration();`	selects the DODAG iteration of which the router is joining
`abstract void selectParents(DODAG theDODAG, int dodagVersion);`	selects, within a given DODAG iteration, those "candidate neighbors"' which are selected as parents
`abstract void selectPreferredParent(DODAG theDODAG, int dodagVersion);`	selects which parent is the preferred one
`abstract void selectSiblings(DODAG theDODAG, int dodagVersion);`	selects the siblings of the router

A subclass of the abstract ObjectiveFunction class, denoted OF0, contains a basic instantiation of the objective function. The rank calculation of a router is based on a hop-count distance from the root, where a router selects its own rank such that it is the minimum rank of all its neighbors plus one. Parents are all neighbors with a lower rank than the router's own rank. The first parent in the list of parents is, then, selected as preferred parent.

14.4 Upward Flow

This section describes how the upward flow mechanism of RPL (*i.e.* DIO and upward data traffic) has been implemented.

14.4.1 DIO

Initially, no router besides the root emits DIO messages. Each DIO message contains the DODAG Configuration option, which lists parameters required by RPL. Upon reception of a DIO, a router that has not yet joined the DODAG, joins it and schedules a timer for starting its own DIO transmissions. When the timer expires, the router selects its parents and preferred parents and then starts emitting DIOs, either using the trickle timer or a periodic timer. The trickle timer follows the specification as described in section 13.6, but does not provide suppression of messages (*i.e.* the redundancy constant k is set to infinity).

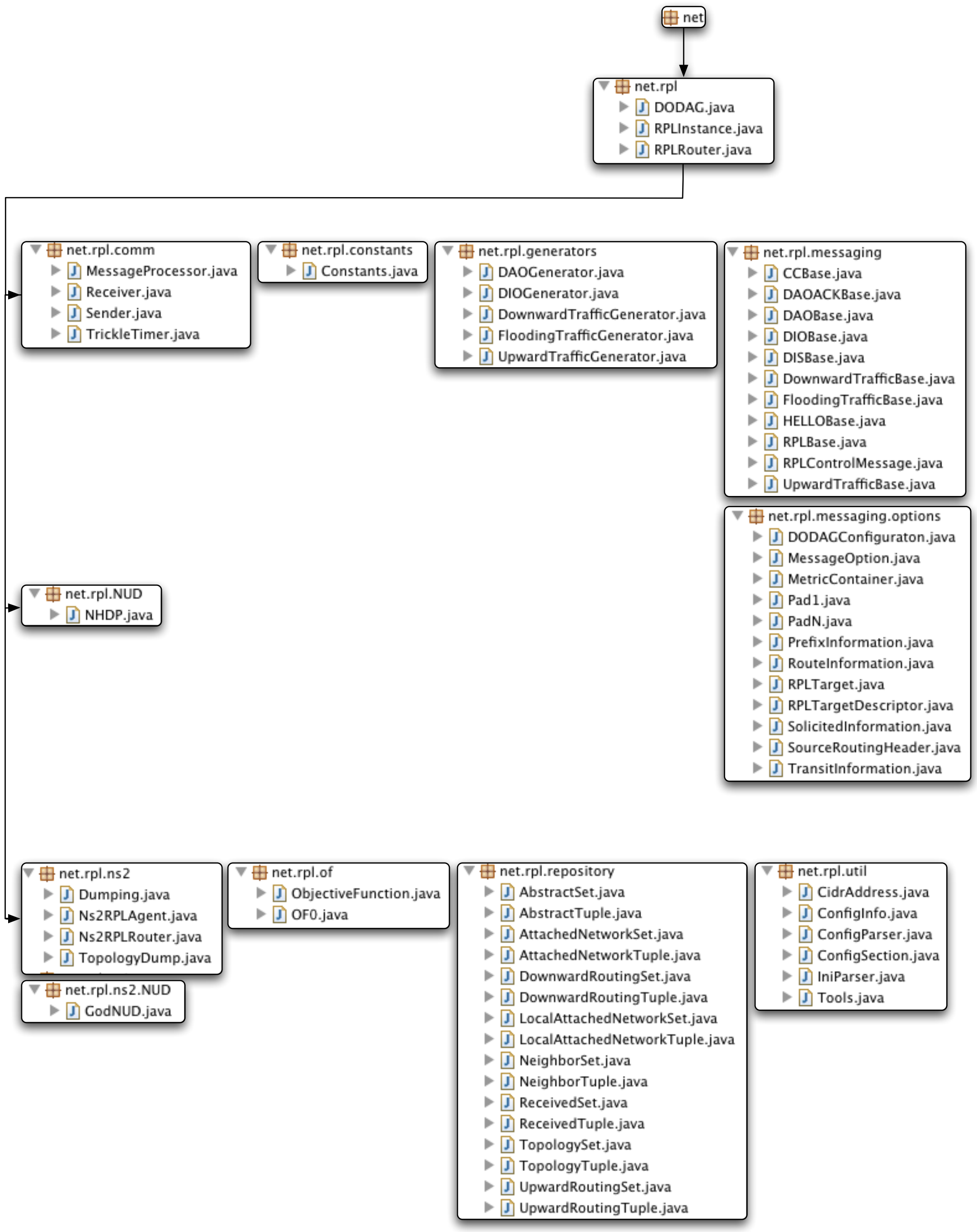


Figure 14.1: JRPL architecture

14.4.2 Upward Unicast Traffic

As JRPL is supposed to run both on real network hardware and in NS2 simulations, and was designed to not require any modifications to the IP stack (*i.e.* to run in user mode), data traffic is encapsulated in RPL messages and forwarded hop-by-hop by JRPL. This allows to run JRPL in user mode (if run on real network hardware), as the forwarding decision is taken directly in the control plane (*i.e.* in JRPL), instead of in the data plane. For NS2 simulations, this implementation choice allows to handle the data traffic forwarding in JRPL and thus in Java, with easy access to all internal data structures of JRPL.

The class `net.rpl.generators.UpwardTrafficGenerator` generates the traffic, with a given interval and message size. An additional message type has been added to the implementation (`net.rpl.messaging.UpwardTrafficBase`), which contains the payload, the source router of the message, a TTL and a sequence number. Unicast traffic is sent up to the root with hop-by-hop transmissions via the preferred parents.

14.5 Downward Flow

This section describes how the downward flow mechanism of RPL (*i.e.* DAO, DAOACK and downward data traffic) has been implemented.

14.5.1 DAO

JRPL implements the non-storing mode of RPL, as described in section 13.3.2.2. Each router advertises the list of its DAO parents, which – in JRPL – only contains the preferred parent of the router. This is conform to [43], which specifies that any subset of the parents of a router can be advertised as DAO parents. While advertising several parents in DAOs allows for multiple downward paths and thus failure tolerance and bandwidth load balancing, using only the preferred parent as DAO parent is less complex, and routing set calculation is a simple spanning tree computation.

Similar to upward traffic, DAOs are sent hop-by-hop via the preferred parents towards the root. The root may then calculate routes to all routers from which it has received DAOs, if it has received DAOs from all routers along the path, depicted in figure 14.2. In the example, two DAOs are sent to the root: router 1 has advertised the root ("R") as DAO parent, and router 3 has advertised router 2 as DAO parent. Unless the root also gets a DAO from router 2, advertising router 1 as parent, it cannot calculate a route to router 3. It may happen that, *e.g.*, the DAO from router 2 is lost, or that it is sent later than the DAO from router 3.

RPL [43] does not specify how to avoid the case where DAOs from routers with a higher rank (*i.e.* "deeper" in the DODAG) arrive before the DAOs from their parents (or in general, DAOs from routers with a lower rank). In JRPL, an implementation choice has been to delay DAO transmissions proportionally to the rank of the router: Once a router has joined a DODAG and selected its parents, it starts emitting DAOs towards the root after `DEFAULT_DAO_DELAY` * `rank_of_router`, such that it is *likely* (but not guaranteed) that the root can calculate a route

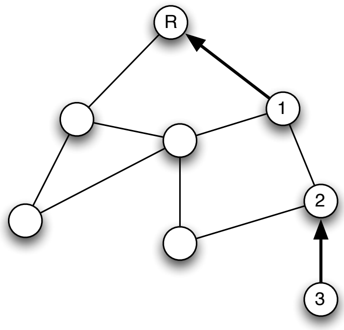


Figure 14.2: DAO route calculation; arrows represent links that are advertised in DAO messages sent to the root (*i.e.* router 1 advertises a link {1–R} in a DAO, and router 3 advertises {3–2})

towards the router from which it receives the DAO. This is in particular important when the root is configured to acknowledge the receipt of the DAO with a DAOACK, as described in section 14.5.2.

As RPL [43] does not specify in which intervals to send DAOs, the implementation choice in JRPL was to send DAOs periodically. Each DAO advertises a link to the preferred parent (which is the only DAO parent in JRPL), with a lifetime of `DAO_REFRESH_INTERVAL`. When the root receives the DAO, it stores the entry for exactly that lifetime, after which the entry expires. "Briefly" before the expiration of the lifetime (where "briefly" is defined as `DAO_REFRESH_SAFETY`), the router refreshes the link by sending a new DAO. As DAOs may be lost, several DAOs are sent in short intervals (denoted `DAO_TIMEOUT`). All the intervals are reduced by a small jitter, according to [34]. JRPL stops sending when `DAO_MAX_TRIES` DAOs have been sent. The timing intervals are depicted in figure 14.3.

The reasons for the implementation choice of the DAO intervals are that (1) DAOs should be received by the root before the routing entry expires, (2) as RPL should accommodate lossy links, sending several DAOs in regular intervals increases probability of their successful reception at the root, (3) limiting the number of DAOs per `DAO_REFRESH_INTERVAL` reduces the required amount of control traffic, and (4) using a jitter reduces the probability of collisions of frames.

14.5.2 DAOACK

If DAOACKs are enabled on the root, each incoming DAO (with the "K" flag set) will be acknowledged by the root by sending a DAOACK message back after a delay of `DAOACK_DELAY`. While [43] does not specify to use the delay – in fact, it does not specify at all how to send or process DAOACKs – adding a delay in JRPL was chosen for two reasons: (1) when many DAOs arrive at the root, each may trigger a routing set recalculation, which may be costly in terms of CPU time (as shown in chapter 8). By waiting a certain time before calculating the routing set and sending back the DAOACK, more DAOs, which can be considered when calculating routes, may have arrived. At expense of some additional delay and a slightly outdated routing

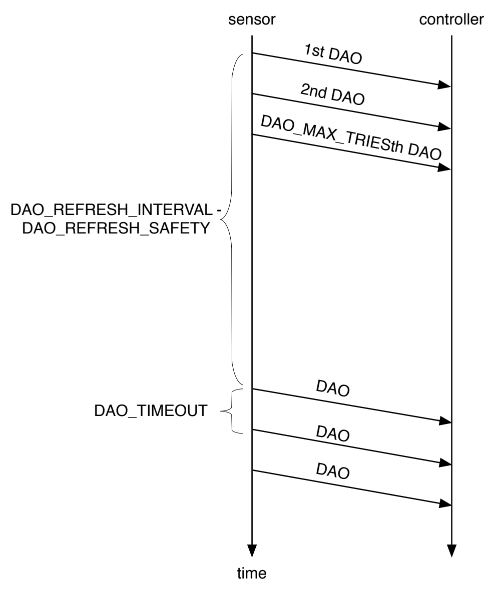


Figure 14.3: DAO intervals

set, fewer routing set recalculations have to be performed. (2) For the reason described in section 14.5.1, it may not be possible to calculate a route to the originator of the DAO, because intermediate routers' DAOs may not yet have been received. By adding a delay before sending the DAOACK, these intermediate DAOs may have been received. `DAOACK_DELAY` should be lower than `DAO_TIMEOUT`, such that the DAOACK is received at the originator of the DAO, before it sends another DAO, as depicted in figure 14.4. If the root does not have a valid route towards the originator of the DAO, it suppresses the DAOACK.

In non-storing mode, the DAOACK, as well as downward traffic (as described in section 14.5.3), is sent to the destination via source routes. As Java does not allow such access to the IP header, nor allows modifications of the IP stack (in order to process source routes), an RPL "source route" message option has been added to the payload of the RPL message. The message option is of similar length as the "IPv6 Routing Header for Source Routes with RPL" [168], which should lead to similar results when evaluating JRPL in NS2. JRPL allows compressing addresses in the source route option, similar to [168] (default is two octets per IPv6 address). At each hop, the router receiving a DAOACK removes its own address from the source route option, until it reaches the final destination (*i.e.* the last address in the message option).

14.5.3 Downward Unicast Traffic

Downward traffic from the root to sensors is encapsulated in RPL messages, similar to upward traffic, as described in section 14.4.2. The traffic messages are sent hop-by-hop using source routes, similar to DAOACK messages, as described in section 14.5.2. If the root has no route towards the destination, the data packet is dropped.

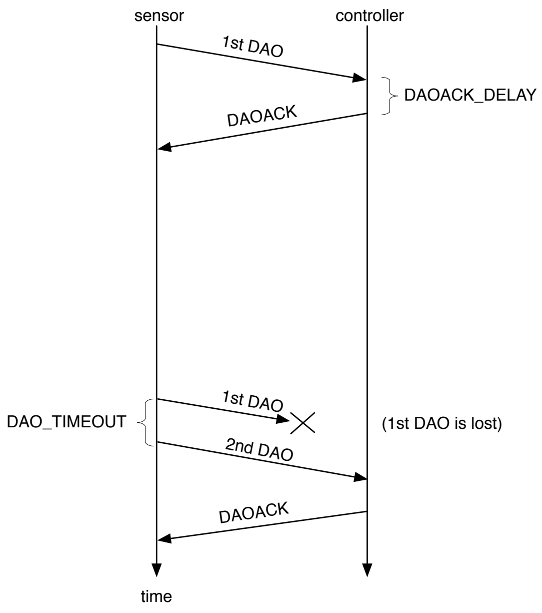


Figure 14.4: DAO intervals with DAOACK

14.6 Local Repair

JRPL provides a local repair mechanism for detecting when a parent becomes unavailable. Two different such "Neighbor Unreachability Mechanisms" (NUD) have been implemented: (1) a "God" mechanism, only working when running JRPL in NS2, which determines that a neighbor is unreachable by checking whether it is still in radio range (based on the position of the node) and has not run out of energy, and (2) a neighborhood discovery mechanism similar to NHDP [37] with periodic HELLO exchange listing all "heard" neighbors, and thus assuring a bi-directional communication. The reason for not including upper layer information (such as missing ACKs) is discussed in section 13.7.

When a router discovers that a parent is not reachable any more, JRPL will update the rank of the parent to infinity, and restart parent and preferred parent selection. The former parent will thus be removed from the parent set. If the parent selection requires the router to increase its rank, the rank will be set to infinity, *i.e.* JRPL applies "poisoning" instead of allowing the router to move deeper in the DODAG, in order to avoid loops (as described in section 13.4). This is conform with [43], according to which the maximum rank increase can be set to 0 (which still allows to increase the rank to infinity). Poisoning will also be used if no parents are left in the parent set.

RPL [43] does not specify how to poison the sub-DODAG of a router. In JRPL, the trickle timer is reset whenever a change in the parent set or the preferred parent is detected by the local repair mechanism. That implies that when a router wants to poison its sub-DODAG, it sends DIOs with a rank of infinity at the minimum interval of the trickle timer[1]. It should be noted

[1]An improvement could be to stop sending DIOs with infinite rank after a bounded number of DIOs, in order

that poisoning the sub-DODAG does not guarantee loop-freeness, as depicted in figure 14.5. In the example, a hop-count based rank is used, the edges between the routers indicate the preferred parent selection. Assuming that router 1 discovers that the root “R” is unreachable, it will poison its sub-DODAG (routers 2 to 5), by sending DIOs with a rank of infinity (“poisoned DIO”). If, however, router 1 receives a DIO from router 2 or 3 before it sends its DIO (or if the poisoned DIO is lost), router 1 may attach to its own sub-DODAG leading to a loop.

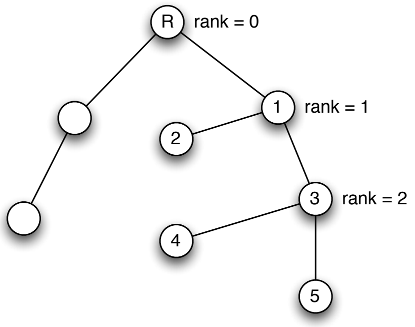


Figure 14.5: A loop between router 1 and 3 may occur if router “R” becomes unreachable, thus router 1 poisons its sub-DODAG, but 3 does not receive the poisoned DIO, and still advertises a rank of 2. Router 1 may then attach to its sub-DODAG with a rank of 3.

One way that situation could be alleviated is to not process any DIOs for a certain time after a router has set its rank to infinity (while continuing to send poisoned DIOs). Thus, the probability is increased that routers in the sub-DODAG have received at least one poisoned DIO, and thus set their rank to infinity as well. This improvement is not implemented in JRPL. However, since loops can occur even with such a mechanism, a loop detection mechanism is implemented in JRPL, as presented in section 14.7.

14.7 Loop Detection

As loops can occur in RPL, such as described in section 14.6, a loop detection mechanism based on data traffic has been implemented, according to the RPL specification. When “upward” data traffic is received on a router, but the previous hop of the packet has a lower rank (according to the neighbor set of the receiving router), the router concludes that there must be a loop. As [43] does not specify how to repair a loop, once it is has been detected, the implementation choice was that the router (1) drops the data packet, (2) sets its own rank to infinity, (3) sets the rank of the previous hop of the data packet to infinity, (4) starts parent and preferred parent selection, and (5) resets the trickle timer.

to avoid unnecessary bandwidth consumption

14.8 Conclusion

JRPL is a Java implementation of RPL, containing the upward and the non-storing downward flow mechanisms. Its architecture and part of the code base, is similar to the JOLSRv2 implementation presented in chapter 7. A basic objective function, based on hop count distances for rank calculation, has been implemented. Local (but not global) repair mechanisms as well as loop detection is integrated.

JRPL provides the core mechanisms of the RPL specification, can be run on both real machines and in NS2 simulations, and is used in the following chapters for evaluation of the RPL protocol and extensions thereof.

Chapter 15

Multipoint-to-Point in RPL

RPL is supposed to operate on large networks, containing "thousands" of constrained sensor routers [43], providing unicast routes from sensors to the root and from the root to sensors. The objective of the specification is to minimize the required state and the control traffic overhead, while maximizing efficiency of the unicast according to the foreseen data traffic flows, as presented in section 13.3.

While some implementations of RPL exist (*e.g.* ContikiRPL [169] and OpenWSN [170]), an extensive simulation and performance evaluation of RPL does not exist, to the best of the author's knowledge. This chapter investigates several performance related parameters of RPL control traffic as well as data traffic in an RPL-based network.

15.1 Chapter Outline

Section 15.2 lists the scenario setting for the performance evaluation of RPL, based on network simulations with NS2. Section 15.3 provides the results of the performance study of the multipoint-to-point performance of RPL. Section 15.4 concludes this chapter.

15.2 Simulation Settings

For the performance evaluation of RPL using an NS2 simulation, the JRPL implementation presented in chapter 14 has been used. The specific settings of the scenarios studied are detailed in table 15.1. For each datapoint, the values have been averaged over 10 runs.

15.2.1 DIO settings

Since routers are not mobile in the simulation, and at the beginning of the simulation, only the root (which is the router with the ID of 0) starts transmitting DIOs. Routers other than the root receiving a DIO start sending DIOs exactly two seconds after no more change in their Neighbor Set has been detected. Each DIO contains the DODAG Configuration option.

Table 15.1: NS2 parameters

Parameter	Value
NS2 version	2.34
Mobility scenarios	No mobility, random distribution of routers
Grid size	variable
Router density	50 / km^2
Communication range	250m
Radio propagation model	Two-ray ground
Simulation time	100 secs
Interface type	802.11b
Frequency	2.4 GHz

The simulations have been performed in two variations:

- with periodic DIO transmission: DIOs are sent periodically with an interval of two seconds minus a jitter of maximum 0.5 s (as defined in [34])
- with a trickle timer: `I_min` is 2 s and `DIOIntervalDoublings` is 20. During the simulation, the trickle timer is never reset.

15.3 Simulation Results

This section describes the results of the NS2 simulation. Figure 15.1 shows an RPL instance of a simulated network with 1000 routers.

15.3.1 RPL Control Traffic

Figure 15.2 shows the maximum and average rank of routers in the DODAG, where the number represents the distance of a router to the root in terms of hops (*i.e.* the maximum rank represents the diameter of the network, the average rank represents the average over all routers). The maximum and average ranks grow logarithmically with the number of routers in the network.

Figure 15.3 depicts the average number of parents of each router in the DODAG. Keeping the density of the network constant with increasing number of routers, the average number of parents grows logarithmically.

Figure 15.4 displays the convergence time of the network, *i.e.* the time that is needed for all routers that are in the same connected component as the root to join the DODAG. Since each router starts sending DIOs two seconds after the last change to its Neighbor Set, the convergence time is roughly two seconds times the maximum rank of the DODAG. The convergence time grows logarithmically with the number of routers in the network.

Figure 15.5 depicts the total control traffic in the network in bytes. The RPL implementation with the trickle timer has significantly less overhead than the periodic timer. The control traffic grows linearly with the number of routers in the network.

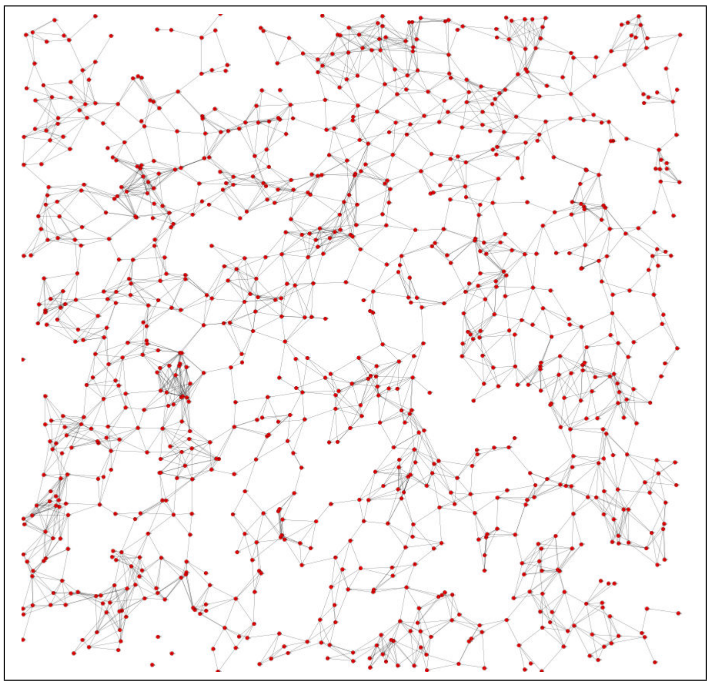


Figure 15.1: Example RPL instance with 1000 routers

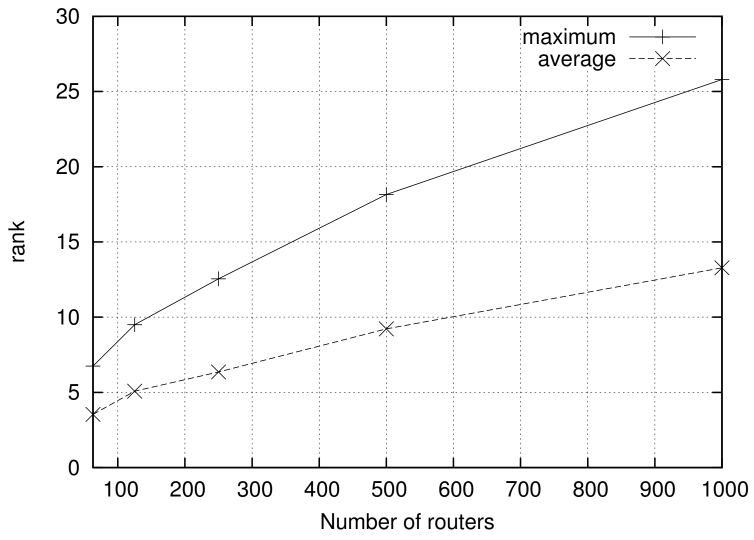


Figure 15.2: Maximum and average rank of routers in the DODAG

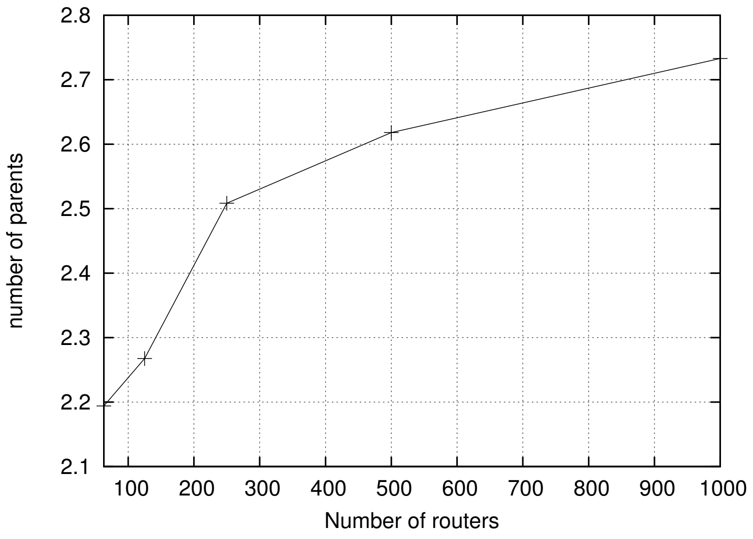


Figure 15.3: Average number of parents per router in a DODAG

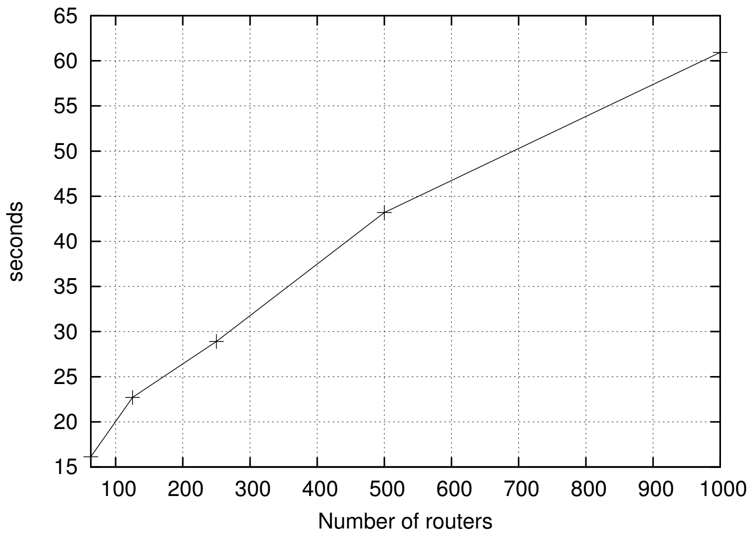


Figure 15.4: Network convergence time

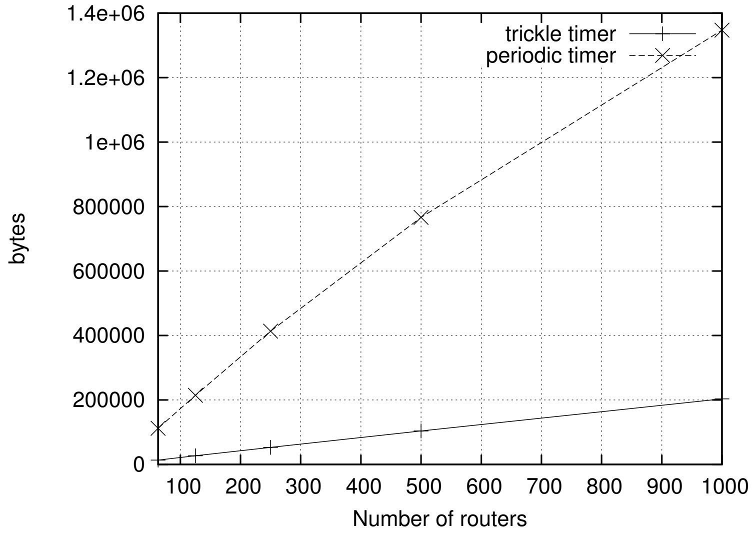


Figure 15.5: Control traffic: overhead in bytes

Figure 15.6 depicts the collision ratio of the DIO messages. Since the RPL implementation using the trickle timer sends significantly fewer DIO messages, the probability of collision is lower.

15.3.2 Unicast Data Traffic

In the following, unicast CBR data streams of 1280 bytes/s have been sent from an arbitrary router to the root, in average five concurrent streams of 10s duration each.

Figure 15.7 depicts the delivery ratio of packets that have arrived at the root. It can be seen that it is constantly high, only few packets are lost due to collisions on lower layers.

Figure 15.8 illustrates the average path length in number of hops that a data traffic traverses before reaching the root. As expected, it grows logarithmically with the number of routers, and is similar to the average rank as depicted in figure 15.2.

Figure 15.9 shows the delay of the data transmission, *i.e.* the time interval from sending the packet at the source until it reaches the destination. Due to the longer path length, the delay increases with the number of routers in the network.

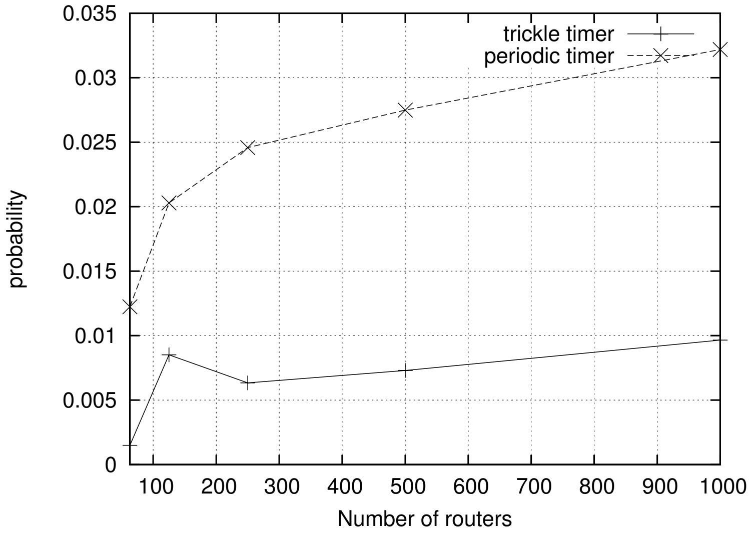


Figure 15.6: Control traffic: collision ratio

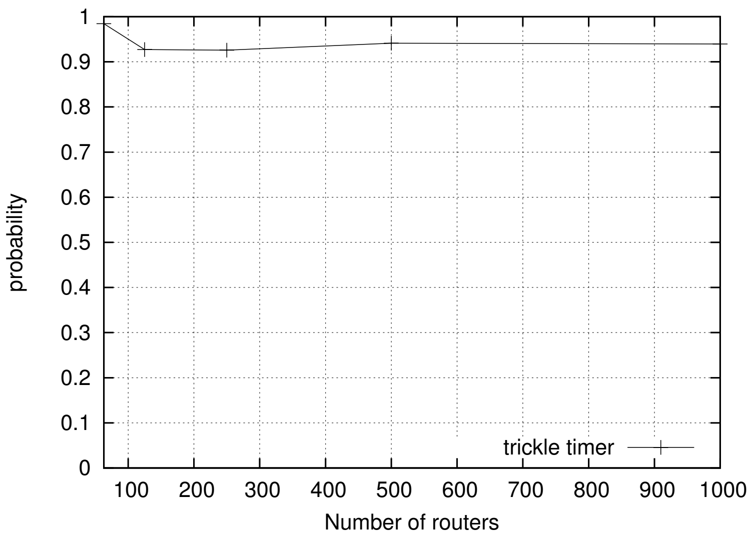


Figure 15.7: Unicast: delivery ratio

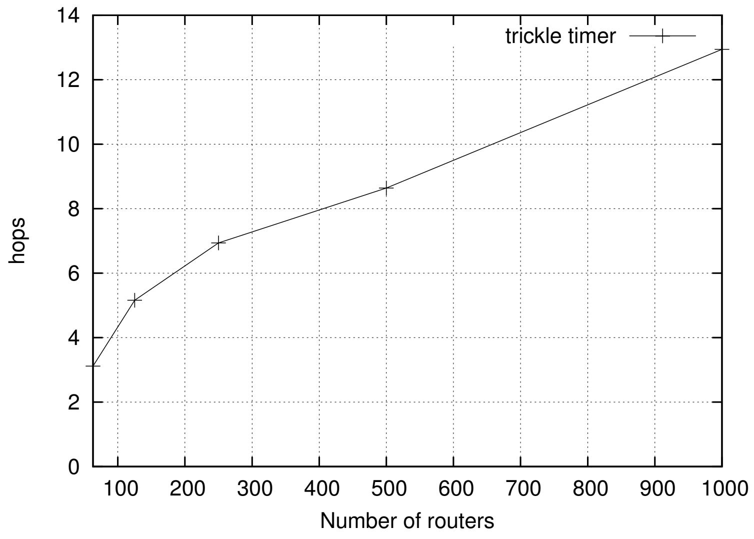


Figure 15.8: Unicast: path length

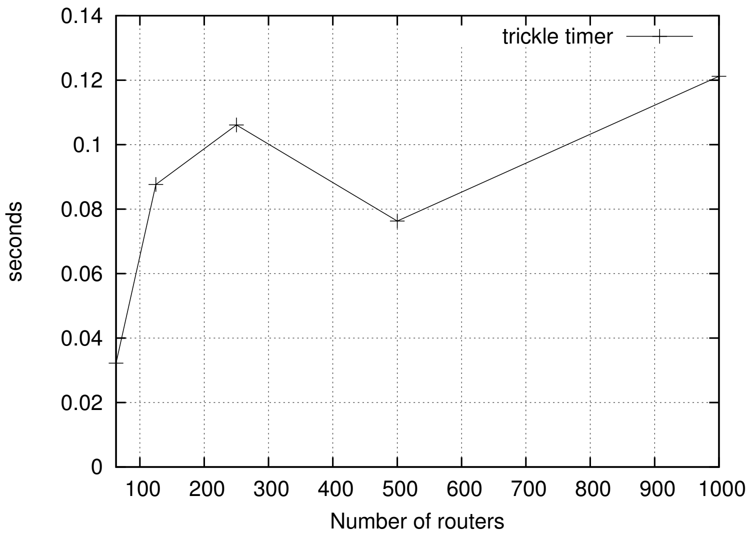


Figure 15.9: Unicast: delay

15.4 Conclusion

The chapter has presented a performance study of the “multipoint-to-point” (sensor-to-root) routing performance of RPL, using network simulations in NS2. This performance study allows to understand the properties, benefits and costs of RPL in Wireless Sensor Networks.

The simulation results show that RPL scales well with the size of the network, in particular with the use of the trickle timer. The control traffic overhead of the protocol is relatively low, and delivery ratio of data traffic is high and stable – even for large networks of several hundred routers.

The contributions of this chapter have been published in [56].

Chapter 16

Broadcast in RPL

The RPL specification [43] does not contain any mechanism for optimized broadcasting data throughout the network. One important application of broadcasting in a WSN is for the root to request that all sensors in the WSN transmit their sensor information – *e.g.* to verify if an alarming condition, signaled by a single sensor, is confirmed by other sensors in the WSN or for bulk-setting configuration parameters in all sensors.

While such a “broadcast” could be accomplished by the DODAG root performing “bulk-unicast” to all sensors in the network, this is hardly efficient due to redundant transmissions of the same packets over the same link. Thus, this chapter investigates ways of providing a reasonable optimized broadcast capability for an RPL routed network, and performs a comparison of the proposed mechanisms based on network simulations in NS2.

16.1 Chapter Outline

Section 16.2 suggests several different mechanisms for providing also support for broadcast traffic in a WSN, by way of using the data structures and topologies already maintained by RPL. Section 16.3 provides a comparative study of the suggested broadcast mechanisms. Section 16.4 concludes this chapter.

16.2 Data Broadcasting in RPL

This section suggests mechanisms for exploiting the DODAG as constructed by RPL in order to undertake better-than-classic-flooding WSN-wide broadcasting. The fundamental hypothesis for these mechanisms is that all broadcast operations are launched from the root of the DODAG. If a sensor needs to undertake a network-wide broadcast, the assumption is that this broadcast is sent to the root using unicast, from where the DODAG root will launch the broadcast operation – this is similar to the basic mechanism for sensor-to-sensor unicast in [43], wherein traffic from the source sensor transits to the DODAG root, for relaying to the destination sensor.

16.2.1 Classic Flooding (CF)

A common baseline for broadcast operations is that of classic flooding: each router relays a broadcast packet upon its first receipt by that router; subsequent receipts of the same packet are suppressed and do not cause retransmissions. This has to its merit that no control traffic is required – however also entails (i) that each data packet must be uniquely identifiable (commonly ensured by embedding a unique sequence number in each broadcast packet, emitted by a given source), (ii) that each router must maintain information (state) for each already received and relayed data packet so as to enable suppression of duplicates, and (iii) each data packet is retransmitted by each router in the network – often with a large degree of redundant transmissions as consequence.

Redundant retransmissions cause increased battery drain, both when transmitting and receiving (and discarding) the redundant packets, and increase contention on the wireless media, increasing the probability of data loss due to collisions. CF is, for these reasons, not suggested as a mechanism for data broadcast in WSNs, but is described here as a baseline for data broadcasting in RPL.

16.2.2 MultiPoint Relay Flooding (MPRF)

MPR, as presented in section 6.3.2, is considered a common improvement over Classic Flooding. In order for MPRF to work, a router must select its MPRs such that a message relayed by these MPRs will be received by all routers two hops away. To this end, each router must maintain, at a minimum, state describing both its neighbor routers, as well as its 2-hop neighbors ("neighbor routers of neighbors"). MPRF – as CF – requires identification of each broadcast packet, and maintenance of state allowing elimination of duplicate packets.

Comparing RPL-specific broadcast mechanisms with MPRF is therefore, to a certain extend, a comparison with "state of the art" of broadcasting in wireless multi-hop networks.

16.2.3 Parent Flooding (PF)

Admitting the RPL "philosophy" of data transmission to sensors originating at (or relaying via) "the DODAG root", RPL lends itself to a first and simple broadcast optimization: restricting a RPL router to retransmit only broadcast packets received from a "parent". Logically, the basic performance hereof should be similar to that of classic flooding: with the broadcast operation initiated from the DODAG root, each router will retransmit the packet upon receipt from a parent. PF does not require any additional control traffic over that which is caused by RPL. PF may apply identification of each broadcast packet, and maintenance of state allowing elimination of duplicate packets in order to avoid multiple retransmissions of the same packet received from different parents – similar to MPRF and CF.

16.2.4 Preferred Parent Flooding (PPF)

In order to not incur any additional in-router state requirements for detecting and suppressing retransmission of duplicate packets, preferred parent flooding utilizes the existing relationship between RPL routers, in order to ensure that no router will forward a broadcast packet more than once. Each RPL router is required to select exactly one preferred parent. Restricting retransmissions of broadcast packets to only those received from the router's preferred parent ensures that duplicates received from other routers (parents or otherwise) are ignored for retransmission.

16.2.5 Preferred Parent MPR Flooding (PPMPRF)

PPF is fundamentally a derivative of the MPRF optimization, attempting further to decrease the number of retransmissions necessary for a network wide broadcast. The idea is as follows: each router, selected as preferred parent, must designate a subset of its "selectees" (children which have selected it as preferred parent) as "preferred children". These preferred children must be selected such that a message, relayed by these preferred children, will reach all its "grand children" – *i.e.* the children of its "selectees".

Whenever a router receives a data packet that is to be broadcast throughout the network, that router will only then forward the packet if (i) at least one parent of that router has selected it as preferred child and (ii) the packet has not been previously received (as determined by a duplicated detection mechanism). It is to be noted that it is not sufficient to restrict forwarding to packets received from the preferred parent of a router, but that packets from *any* parent of that router have to be forwarded if the router has been selected as preferred child by *at least one* of its parents. The rationale is illustrated in figure 16.1. Assuming that the root of the network (router *0*) has selected B as preferred child as indicated by the downward arrow, and B forwards a packet originating from the root. If forwarding was restricted to packets received from the preferred parent of a router, D would not forward the packet from B (since it is no preferred parent of D), and thus X would never receive the packet.

Compared to "classic" MPR selection, the preferred children selection (i) concerns only coverage of "grand children" (*i.e.* "downward" in the DODAG as constructed by RPL) and (ii) is restricted by the preferred parent selection from RPL.

This restriction entails less liberties with respect to selecting relays for "best 2-hop coverage". It is quite possible that the child providing the "best" coverage of a router has not selected that router as preferred parent, and that therefore PPMPRF will result in more relays than MPRF. In RPL, the preferred parent selection is intended to optimize for "best upwards paths towards the DODAG root" (possibly according to some deployment specific optimization criteria), which may not coincide with what would be optimal for "best downwards coverage".

The PPMPRF mechanism also requires that each router knows (i) which children have selected it as preferred parent (*i.e.* its *selectees*), and (ii) which routers are preferred children of these selectees. This information can be made available through adding an option to DIO messages, emitted by all routers running RPL.

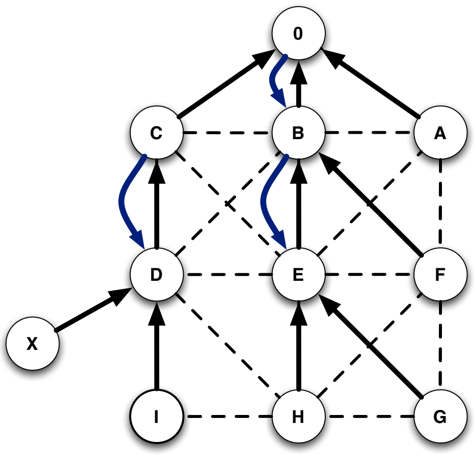


Figure 16.1: PPMPRF: Example showing the need to forward packets not only received by the preferred parent, but by any parent if the router is selected as preferred child by at least one of its parents. Upward arrows depict preferred parent selection, downward arrows preferred child selection.

16.2.6 Optimized Preferred Parent MPR Flooding (PPMPRF-opt)

This mechanism represents a small optimization over PPMPRF, in that it provides all neighboring routers with the same rank with information, encouraging coordinated preferred parent selection so as to try to reduce the number of routers selected as preferred parent. Thus, a router will select as its preferred parent among its parents, the one which most of its adjacent routers also have in their parent set. Given a tie, the parent which a majority of the adjacent routers have already selected as preferred parent will be chosen. Thus, in addition to the information indicated for PPMPRF, PPMPRF-opt requires all parents to be advertised.

Figure 16.2(a) depicts an example of preferred parent selection, as may happen in basic RPL: a router selects its preferred parent amongst all its parents with the lowest rank in an uncoordinated way. Worst case (in terms of redundant transmissions and therefore possible collisions when broadcasting), routers D, E and F all select different preferred parents (C, B, and A respectively). Similarly, I, H, and G may select three different preferred parents. For PPMPRF, this means that all routers, other than 0, will be selected as MPRs and thus retransmit a broadcast.

Figure 16.2(b) depicts a coordinated preferred parent selection. Router D will advertise all its parents (C and B) in its control messages, as will E (parents C, B and A) and F (parents B and A). D has an equal choice between parents C and B, and F has the same choice between B and A. E will select B as preferred parent because this is the only parent that both of its adjacent routers can also select as preferred parent. Once D and F receive a control message from E, advertising that B is selected as preferred parent, they will also select B. Thus, only routers B, E and H will be selected as preferred parents and therefore retransmit a broadcast.

Such coordinated preferred parent selection may be a double-edged sword for RPL. While it

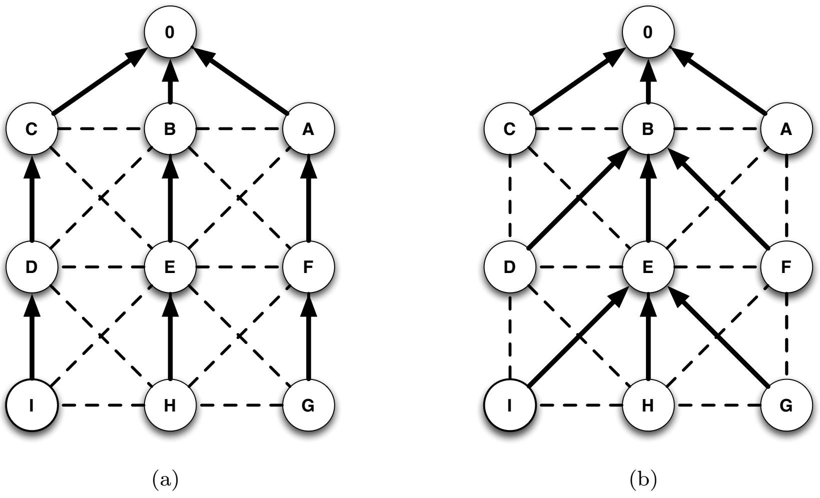


Figure 16.2: Uncoordinated PP selection (a) and coordinated PP selection (b) in the same network. Solid arrows indicate the selection of a parent as preferred parent; dotted lines the connectivity of the network.

is a potential benefit for broadcast traffic from the DODAG root, unicast traffic flows towards the DODAG root via preferred parents. Thus, coordinated selection of preferred parents implies that unicast traffic is concentrated through a subset of the routers in the network, possibly increasing congestion in these routers, increasing the battery drain in these routers etc.

16.3 RPL Performance Study

This section presents results of a simulation study of RPL with the NS2 simulator. Several properties of the DIO mechanism, as well as unicast and broadcast data traffic have been evaluated.

16.3.1 Simulation Settings

The specific settings of the scenarios studied are detailed in table 16.1. For each datapoint, the values have been averaged over 10 runs.

16.3.2 Broadcast Data Traffic

In the following, the broadcast mechanisms presented in section 16.2 are analyzed in terms of MAC layer collisions, delivery ratio, overhead, delay, and path length. CF and PF (without duplicate detection) are not considered since their performance is expectedly much worse than any of the other mechanisms.

Figure 16.3 depicts the number of collisions of frames on the MAC layer, for the different broadcast mechanisms. MPRF and PPMPRF-opt yield the lowest number of collisions among the analyzed protocols, with PPMPRF causing about the same number of collisions as PF+DD

Table 16.1: NS2 parameters

Parameter	Value
NS2 version	2.34
Mobility scenarios	No mobility, random distribution of routers
Grid size	variable
Router density	50 / km^2
Communication range	250m
Radio propagation model	Two-ray ground
Simulation time	100 secs
Interface type	802.11b
Frequency	2.4 GHz

(PF with duplicate packet detection). This is expected, as in MPRF, relays are explicitly selected so as to avoid redundant retransmission by topologically close routers, and the coordinated preferred parent selection in PPMPRF-opt also reduces the number of relays. PPMPRF without coordination entails more relays, as more routers in the network will be selected as preferred parents, which in turn select the relays (*i.e.* preferred children). In PPF, topologically close routers are likely to have chosen the same preferred parent and so will explicitly produce redundant retransmissions. Consider the example in figure 16.4, wherein a broadcast transmission is made by router 0 and relayed as indicated by the solid arrows. In PPF, as indicated in figure 16.4(a), each router will select its preferred parent and retransmit the packet once upon receipt from that preferred parent. Routers A, B and C all receive the transmission directly from router 0. Routers D, E and F have all chosen one of A, B and C as preferred parent and will thus all retransmit when receiving the transmission from their chosen preferred parent – similar for I, H, G, even though these three do not have any routers further down the network. In contrast, in figure 16.4(b), MPRs have been selected. Router 0 has selected B as MPR (as B "covers" D, E, F) and router B has chosen router E as MPR (as it covers all of G, H, I). As there are no further routers "below" in the network, router E has chosen no MPRs downwards. Thus, only B and E retransmit the broadcast packet from 0 – *i.e.* for each "level" in this simple network, only a single transmission occurs, with no collisions at each level.

Figure 16.5 depicts the delivery ratio of broadcast packets. The delivery ratio of the MPRF and PPMPRF-opt mechanisms are the highest of the compared broadcast mechanisms, with PPMPRF being not much below MPR. This can be interpreted as a tradeoff between redundancy and efficiency: in relatively scarce networks (such as the simulated scenario) a higher redundancy of relays, such as in PPMPRF, can lead to a higher delivery ratio, despite of the increased number of collisions, as observed in figure 16.3. In dense networks, however, the large number of collisions with more redundant delays can reverse that effect and reduce the delivery ratio. A detailed analysis of the MPR relaying mechanism can be found in [106].

PF+DD has a higher delivery ratio than PPF, due to the redundancy of transmissions – when a router receives the same broadcast packet from several of its parents, chances are higher that at least one of the packets will reach the router, while if the one transmission from the preferred parent in PPF is lost due to a collision, the router will not forward the other incoming packets

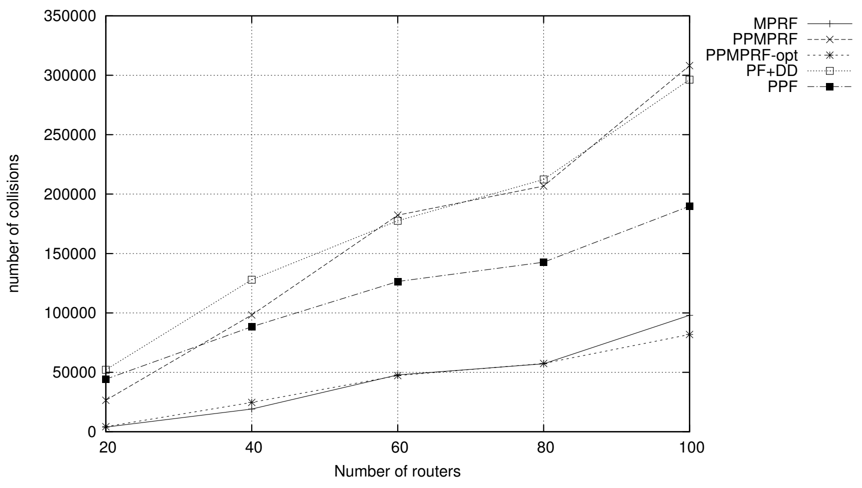
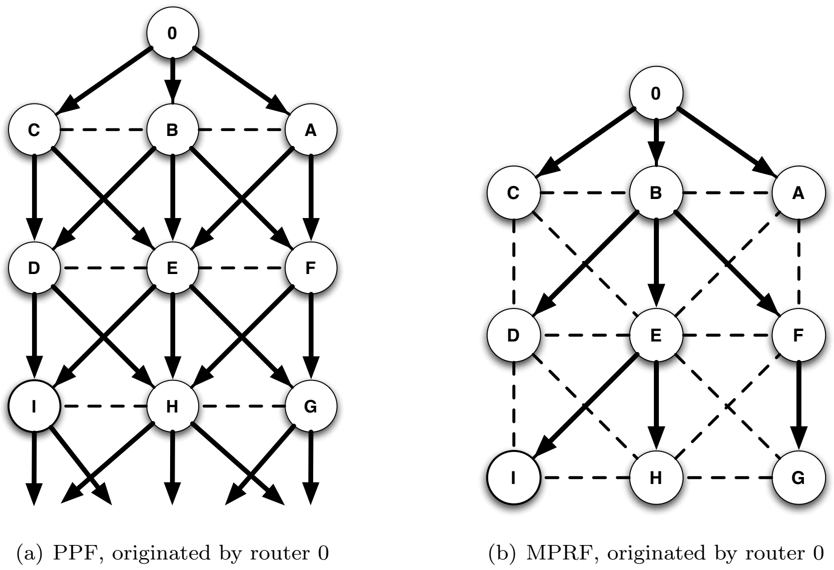


Figure 16.3: Broadcast: total number of MAC layer collisions



(a) PPF, originated by router 0

(b) MPRF, originated by router 0

Figure 16.4: PPF (a) and MPRF (b) in the same network. Solid arrows indicate transmission of a packet; dotted lines the connectivity of the network.

from its (non-preferred) parents. The higher delivery ratio of PF+DD is at the expense of vastly higher media load, as depicted in figure 16.6: the cumulative number of bytes transmitted during the simulations are significantly higher for PF+DD and for PPMPRF without the optimization. PPF incurs a lower overhead than PF+DD with MPRF still outperforming PPF by a large, and constant, margin. PPMPRF-opt has a similar overhead as MPRF.

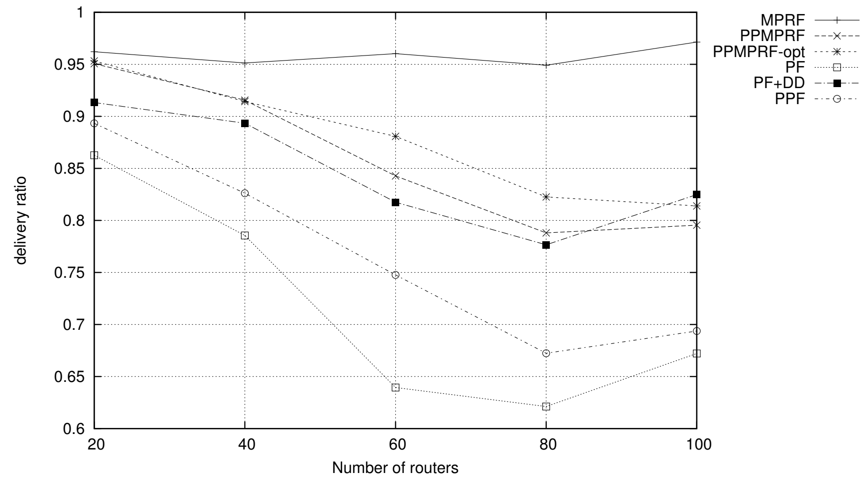


Figure 16.5: Broadcast: delivery ratio

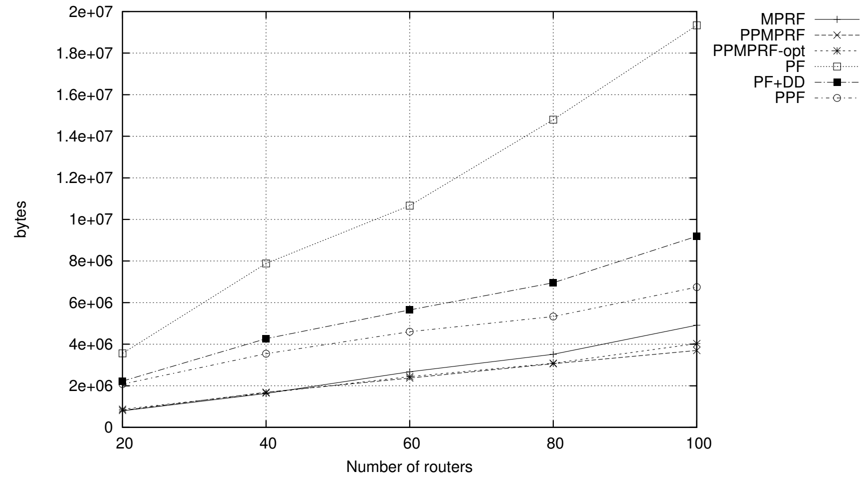


Figure 16.6: Broadcast: total retransmission overhead

Figure 16.7 depicts the average end-to-end delay for data traffic from the root to every WSN router in the network, and figure 16.8 depicts the average path length of successfully delivered data packets. The optimized MPR-based broadcast mechanisms incur the lowest delay of the protocols, while PPF causes a slightly lower delay than does PF+DD. The, on average, longer path lengths of MPRF are due to the data delivery ratio being higher – MPRF successfully "reaches" routers farther away from the root (as depicted in figure 16.9). It has been shown ([106])

that MPR leads to optimal path length. That means that every mechanism indicating a shorter path in the figure entails a lower reachability of routers further away from the broadcast source. Longer paths indicate suboptimal paths. It is worth observing that MPRF achieves the optimal path length with a lower delay still. This can in part be explained by the fact that MPRF ensures that data is flooded via shortest paths, and in part by the fact that with fewer retransmissions, less media and queue contention occurs.

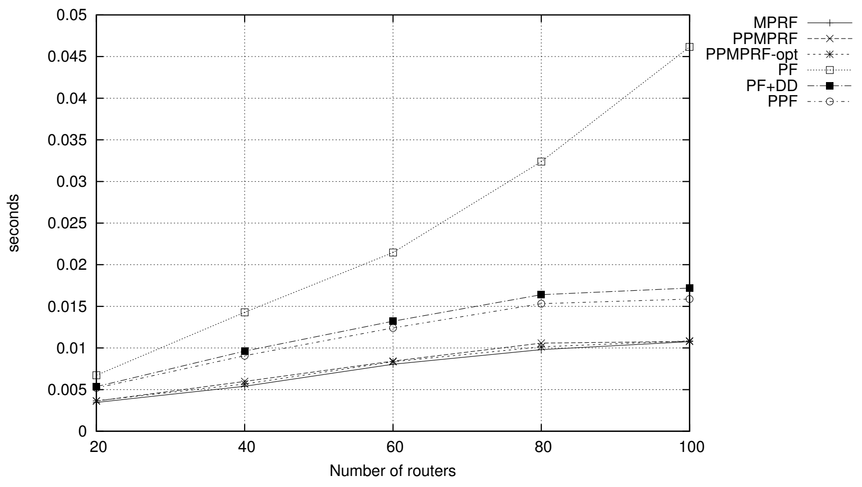


Figure 16.7: Broadcast: average delay

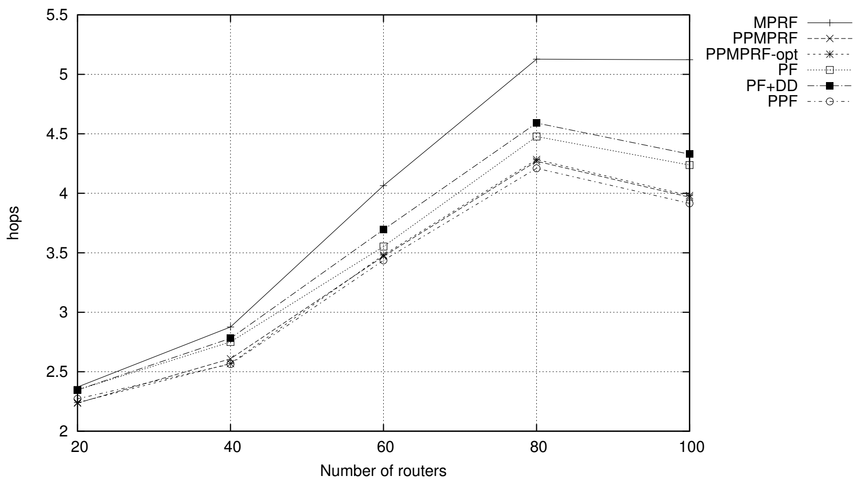


Figure 16.8: Broadcast: average path length

16.3.2.1 PPF with Jitter

In the results presented in section 16.3.2, data traffic has been promptly forwarded by each WSN router, without explicit delay. As has been shown in [171, 34], adding a random jitter before

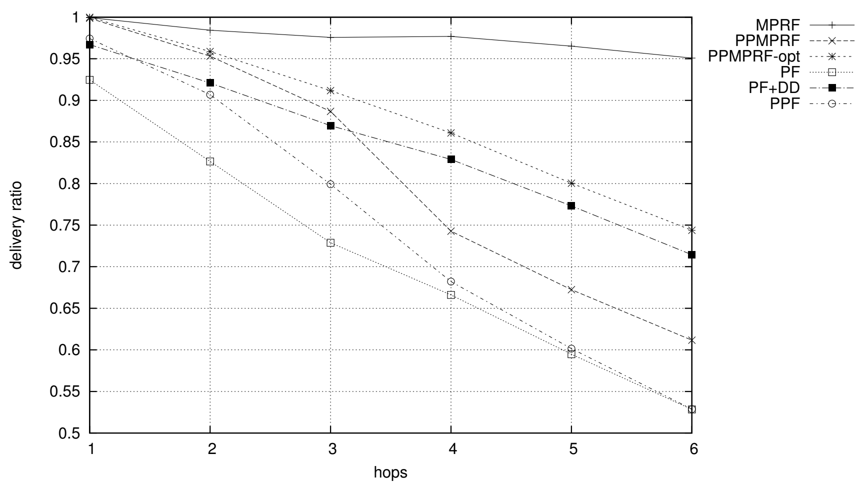


Figure 16.9: Broadcast: traffic delivery ratio with respect to distance from the root in hops (with 100 routers in the network)

retransmitting a broadcast packet can significantly reduce the number of collisions and, therefore, increase the delivery ratio for broadcast packets. In the following, the effect of adding jitter to PPF is investigated.

Figure 16.10 depicts the collision ratio of frames when using no jitter, and a random jitter uniformly distributed between 0 and 500 ms respectively. With jitter, the collision ratio is much lower than it is without. This is due to the fewer concurrent retransmissions by adjacent WSN routers. Comparing to figure 16.3, PPF with jitter yields a collision ratio comparable to, or lower than, MPRF without jitter.

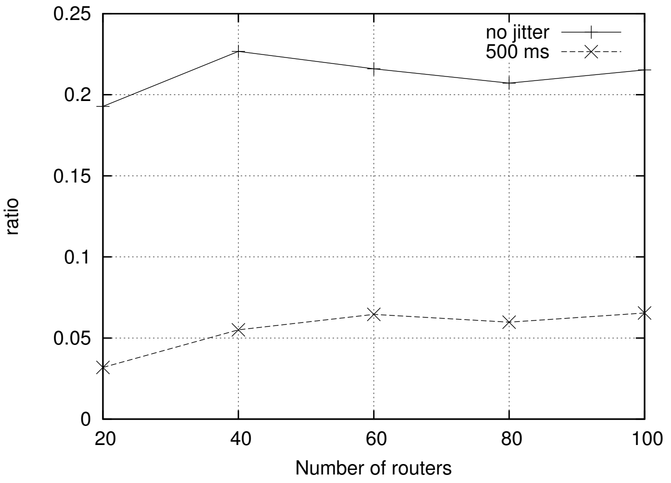


Figure 16.10: Collision ratio of PPF with jitter

As a consequence of the lower collision ratio, the delivery ratio of PPF with jitter is higher than it is without, as depicted in figure 16.11. Comparing to figure 16.5, the delivery ratio of

PPF still remains consistently below that of MPRF, even when PPF is used with jitter.

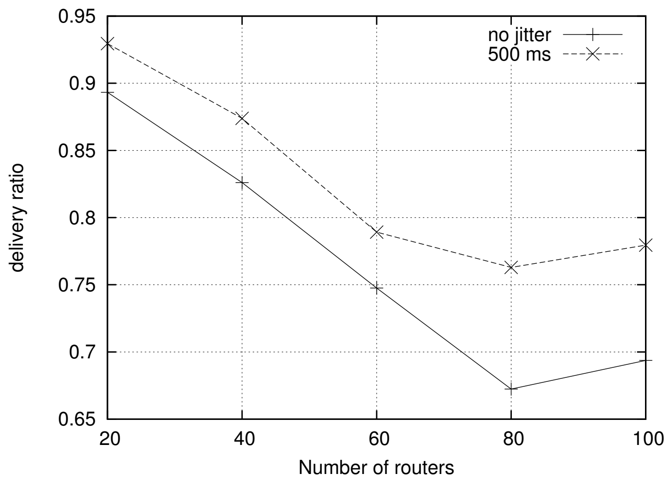


Figure 16.11: Delivery ratio of PPF with jitter

The drawback of using jitter is a higher end-to-end delay of packets, as depicted in figure 16.12. With jitter, the delay is considerably higher than it is without.

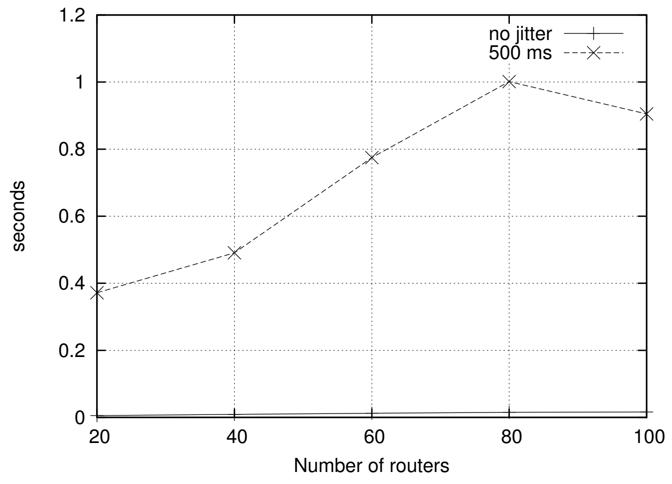


Figure 16.12: Average delay of PPF with jitter

16.4 Conclusion

The unicast routing protocol RPL does not specify any broadcast mechanism, which allows to flood information from the root to all other routers in the network. One important application of broadcasting in a WSN is for the root to request that all sensors in the WSN transmit their sensor information – *e.g.* to verify if an alarming condition, signaled by a single sensor, is confirmed by other sensors in the WSN or for bulk-setting configuration parameters in all sensors.

This chapter has presented several optimized broadcasting mechanisms, using the rooted DAG-like logical structure, maintained by RPL. A comparative study of these broadcast mechanisms has been conducted, showing their performance using different metrics.

These broadcast mechanisms, denoted "Parent Flooding" (PF), "Preferred Parent Flooding" (PPF) and "Preferred Parent MPR Flooding" (PPMPRF) adhere to the "root-oriented" concept of RPL, in that all broadcast operations are to be initiated by the root of the DAG. PF, PPF and PPMPRF are studied and compared by way of network simulations – and as a point of comparison, MPR Flooding (MPRF), is also subjected to the same network scenarios in the simulator.

The results show that the MPR based broadcast mechanisms lead to a much lower bandwidth consumption and a higher delivery ratio than PF and PPF, with the expense of additional control traffic information that is required. If such information is not provided, PPF leads to a better performance than PF. In order to further reduce collisions in the network, using a jitter (*i.e.* a delay when forwarding a message on a router) is evaluated: the jitter decreases the collisions, and thus increases the delivery ratio, but also increases the end-to-end delay.

The contributions of this chapter have been published in [57, 58].

Part V

Conclusion

An ad hoc network is, essentially, a collection of – possibly mobile – routers, communicating among themselves over wireless links and thereby forming a dynamic, arbitrary and often multi-hop graph. Core properties of ad hoc networks are their autonomy, *i.e.* that they can be deployed with no a priori configuration, and their ability to cope with a high degree of network topology dynamism (often due to the mobility of routers, although a dynamic network topology may be due to factors other than mobility).

In the early 2000s, ad hoc networks were mainly deployed as testbeds in research facilities. For logistic and financial reasons, the size of these testbeds is often limited, and as the testbeds are typically under control of researchers, properties of the network (such as traffic patters, topology, routing protocol and deployed applications) do not necessarily correspond to the behavior of "real-world" deployments. Apart from such research testbeds, public community ad hoc networks, such as the FunkFeuer network in Vienna, have grown to considerable sizes outside the confines of research. The FunkFeuer network comprises several hundred routers, mounted on rooftops of buildings in Vienna, enabling data exchange between the users of the network, as well as providing a connection to the Internet.

In order to allow such ad hoc networks to grow further, new protocols and solutions have to be developed, as the autonomy and high topology dynamism renders current Internet protocols unsuitable for ad hoc networks. To that effect, the Internet Engineering Taskforce (IETF) has tasked several Working Groups with specifying protocols for different challenges of such networks: *e.g.* routing, automatic configuration, management, and security. While initially specified protocols (such as OLSR and AODV) served their purpose of establishing communication between ad hoc routers, they proved to be inflexible for protocol extensions. Therefore, new protocols are required to solve these problems. This manuscript analyzes the behavior and evaluates the performance of several of the protocols being specified as of 2011 (such as OLSRv2 and RPL) and proposes extensions thereof for their capability of allowing ad hoc networks to be (1) more performant, (2) to scale well, to (3) be integrated into the Internet, and (4) to be deployed outside of research testbeds.

In this manuscript, a software framework is proposed which allows to run Java routing protocols within the network simulator NS2. The preexisting tool AgentJ allows for using Java agents on an NS2 node, but was not capable of instantiating routing agents. A modification of AgentJ is presented, which enables the use of Java routing protocols in NS2 without modification of the implemented Java routing protocol – adhering to the Java slogan "write once, run everywhere"– by rewriting bytecode when loading the Java classes. Java eases prototype development of protocols for research purposes, as it provides many predefined classes (*e.g.* hash tables, trees) and does not require manual memory management. Moreover, once AgentJ has been installed, NS2 does not need to be recompiled when a new Java routing protocol implementation is added. The routing functionalities have been included in the stable distribution of AgentJ.

A precondition for any routing protocol is that routers have unique and topologically correct IP addresses assigned. To that effect, the IETF has assigned the AUTOCONF Working Group in 2005 to specify autoconfiguration protocols for ad hoc networks. However, no single protocol has been standardized as of 2011. This manuscript provides a detailed architectural discussion of ad hoc networks, which explains the problems the AUTOCONF Working Group has had, and compares the ad hoc network architecture with the architecture of the Internet, which can be considered as a network which scales well. The manuscript suggests an architectural model which allows to seamlessly integrate ad hoc networks into the wider Internet, and to scale such networks. This work has contributed to the AUTOCONF Working Group, as part of an "IP Addressing Model in Ad Hoc Networks" (RFC5889).

In order to be able to deploy large networks, automatic configuration of addresses and prefixes on ad hoc routers is necessary, for avoiding manual configuration of thousands of routers. Therefore, an autoconfiguration protocol, which is based on the proposed architectural model, is presented in this manuscript. The protocol verifies uniqueness of a temporary prefix within the network, and does not rely on an underlying routing protocol, but instead uses locally unique identifiers (UUIDs). The behavior of the protocol is formally verified by means of model checking. Several extensions and optimizations of the protocol are discussed, which allow to reduce the necessary overhead of the algorithm. The proposed algorithm is one of the first specified autoconfiguration solutions that adheres to the address architecture model that has been specified in the AUTOCONF Working Group, and which has been proved to be correct and performant.

Once IP addresses and prefixes are assigned, a routing protocol provides connectivity between routers in the ad hoc network. OLSRv2 enables to scale ad hoc networks to large sizes, in particular due to the Multipoint Relay (MPR) mechanism. This manuscript investigates performance and scalability of OLSRv2 and extensions thereof in different deployments. A limiting factor of any routing protocol is the in-router resource requirement of the protocol (CPU, memory). This applies in particular for constrained routers in ad hoc networks. This manuscript proposes an integration of a dynamic shortest path (DSP) calculation in OLSRv2, allowing to scale OLSRv2 orders of magnitude better (in terms of in-router resource requirements) compared to a classic Dijkstra, while keeping compatibility with the OLSRv2 specification. Moreover, it is shown that the particular characteristics of ad hoc networks make it even more suitable for such a dynamic shortest path algorithm than classical, wired networks. These results demonstrate that it is possible to scale ad hoc networks to large sizes, and therefore deploy this kind of networks outside of limited-size testbeds.

Another optimization in ad hoc networks, which is presented in this manuscript, is external management and monitoring of ad hoc routing protocols. By allowing to remotely access routers and to control parameters of the routing protocol, a management framework facilitates to "tweak" parameters leading to, for example, a more stable perceived topology and to a lower control traffic overhead, and therefore to a higher delivery success ratio of data packets, a lower end-to-end delay, and less unnecessary bandwidth and energy usage. Such optimizations improve the scalability of networks to a large number of routers. This manuscript proposes an SNMP-based framework to manage and control performance related objects on routers running OLSRv2. The framework has also been proposed to the IETF, and is in final phases of standardization by the IETF in 2011.

Moreover, the manuscript evaluates the performance of SNMP in OLSRv2-based ad hoc networks, using the previously presented management framework. While SNMP is sometimes considered too "heavy-weight" for ad hoc routers, as of 2011 there is no standardized "light-weight" management protocol, and therefore SNMP remains the prevailing management protocol for routers. This performance study of SNMP in ad hoc networks is the first of its kind, and thus allowing to understand performance metrics of the protocol in ad hoc network deployments.

In some ad hoc network deployments, with "unstable" links and topologies, it may be preferable to accept a higher delay of datagram delivery by retransmitting the data later for temporarily unavailable destinations of IP datagrams, instead of dropping them ("delay tolerant networking"). This manuscript tests such delay tolerant behavior in OLSRv2-routed ad hoc networks by means of network simulations in increasingly large ad hoc network deployments. In order to understand the effect of delay tolerant routing in "extreme" cases, a special network model is proposed, which leads to a weak performance of routing protocols without delay tolerant routing, whereas high delivery ratios can be achieved – at expense of a higher delay – if datagrams are buffered and scheduled for later transmission. Several different mechanisms for buffering packets are proposed and compared, and it is demonstrated that these mechanisms lead to delivery ratios significantly higher than standard OLSRv2 in the given scenarios; moreover, it is shown that this is close to the upper bound.

Security threats become a major concern in today's computer networks. Network integrity in routed networks is largely preserved by physically controlling access to the communications channel between routers, which allows some means of protection to classical, wired networks. In an ad hoc network, often operated over wireless interfaces, this is less obvious: physical access to the media between routers is not delimited by a cable, but is available to anyone within transmission range. For this reason, securing ad hoc networks becomes a necessity. This manuscript presents a detailed analysis of security threats to OLSRv2, by taking an abstract look at the algorithms and message exchanges that constitute the protocol, and for each protocol element identifying the possible vulnerabilities and how these can be exploited. The presented disruptive attacks to OLSRv2 are classified in several categories. For each, it is demonstrated, whether OLSRv2 has an inherent protection against the attack. This work is the first detailed security analysis of OLSRv2, setting the stage for a protection mechanism for OLSRv2.

Based on the vulnerability analysis, a logical router and link admittance control mechanism for OLSRv2 is specified, which alleviates a large number of the proposed attacks and which preserves the network integrity of ad hoc networks, similar to the "cable" in wired networks. The mechanism allows a router to verify each advertised link from incoming control messages, by signing "both ends of the link". The router and link admittance control protocol extension is generic, in that it is not tied to any specific cryptographic system. Indeed, the mechanism operates as long as the choice of cryptographic system allows for per-principal authentication and signature generation. A performance study of this extension is presented, quantifying the impact in terms of increased control traffic overhead and increased per-message generation and processing time, exemplified by using RSA and ECDSA. Part of the specification of the security mechanism has become a Working Group draft in the IETF and is likely to be standardized in 2011.

Finally, the manuscript investigates scalability of wireless sensor networks (WSNs) – networks consisting of thousands of constrained routers whose primary task is to acquire data through attached sensors and to send that data to a central controller. This manuscript evaluates a routing protocol for such WSNs being specified by the IETF as of 2011, denoted "IPv6 Protocol for Low Power and Lossy Networks" (RPL). A critical discussion of the protocol is provided, as well as a performance evaluation for unicast data traffic. This work presents one of the first detailed simulation evaluations of the RPL protocol, which is considered as *the* standard protocol for WSNs (by the ROLL WG in the IETF). While the results show that for "upward" traffic (sensor-to-controller), RPL is efficient in terms of control overhead, the protocol is complicated to implement, it does not provide mechanisms to detect asymmetric links, the "downward" mechanism is complex with many changes during the specification process, and seems less well understood than upward data flows.

RPL does not contain any mechanism to efficiently flood data from the central controller to all other routers. One important application of broadcasting in a WSN is for a controller to request that all sensors in the WSN transmit their sensor information – *e.g.* to verify if an alarming condition, signaled by a single sensor, is confirmed by other sensors in the WSN or for bulk-setting configuration parameters in all sensors. This manuscript presents several optimized broadcasting mechanisms, using the structures already maintained by RPL. A comparative study of these broadcast mechanisms is conducted, showing that using MPR-based mechanisms – similar to those used in OLSRv2 – leads to an efficient network-wide broadcast.

Future Work

Several extended studies of the presented protocol extensions could be pursued, some of which are discussed in the following.

The autoconfiguration protocol, presented in chapter 5, works independently of any routing protocol. However, if a routing protocol is used in the network, then additional information from that routing protocol could be used to reduce the control traffic overhead of verifying uniqueness of the tentative prefixes. In particular, link state routing protocols, such as OLSRv2, provide full topology information, and could therefore be efficiently applied to limit the overhead of the otherwise required classic flooding in the protocol. If a "defensive" router, which has already been configured a unique prefix, runs such a routing protocol, it could directly verify in its routing table whether the requested tentative prefix of a configuring router exists in the table, instead of flooding the request through the network. Such an extension could be specified and compared to the basic protocol.

Moreover, even if no routing protocol is applied, optimized flooding techniques (such as MPR flooding or multicast using the Trickle algorithm) could be used instead of classic flooding. A comparison of such extensions would likely show a performance improvement over the basic algorithm. Finally, it could be investigated how the presented autoconfiguration protocol could be extended or modified in order to guarantee unique prefixes with partitioning and merging networks.

As for the presented OLSRv2 extensions, several further studies could be pursued:

The dynamic shortest path algorithm, presented in chapter 8, facilitates to scale the routing table calculation to large networks. However, another problem in large networks still has to be considered: the control traffic overhead in large networks may (i) exhaust the wireless channel capacities, and (ii) the processing and generation of a large quantity of control messages on each router may be a CPU-intensive task. A mechanism that reduces the control traffic overhead, together with the presented efficient routing table calculation, would enable to scale ad hoc networks further.

The network management study, presented in chapter 9, focuses on the SNMP protocol. While the study has shown that the control traffic overhead of SNMP in certain scenarios can be low, the complexity of the protocol – and therefore the code size of the SNMP implementation – may consume too much memory on the routers. Specifying a simpler protocol for management of ad hoc networks would reduce the code size and may therefore be more applicable for constrained routers. Such a protocol could also allow management of whole *networks* instead of each single router separately. A future work could compare such a new protocol in its behavior compared to SNMP.

As for the delay tolerant networking (DTN) mechanism for OLSRv2, presented in chapter 10, a further study could evaluate the effect of such a DTN mechanism on upper-layer protocols, both on transport layer and application layer. It is possible that due to the higher delay caused by the mechanism, upper-layer protocols need a particular parametrization, or that some protocols may not work correctly at all. Understanding the effects of the increased delay on these protocols would help to deploy the DTN mechanism in real-world deployments.

Concerning the security extensions, presented in chapter 12, one assumption of the mechanism is that credentials (*i.e.* cryptographic keys) are distributed and revoked by some external mechanism. Understanding how an efficient mechanism for key distribution could be deployed together with the presented signature mechanism, would provide a complete admittance system for OLSRv2-based networks.

A major future work is to consider link metrics in OLSRv2, in terms of performances studies of OLSRv2 with different link metrics, for an extended security evaluation, for the DTN mechanism and for other extensions in OLSRv2. This manuscript has not considered link metrics, because in its current revision, the OLSRv2 specification does not yet include the link metrics mechanism. It is, however, intended to be included in the following revision of the draft.

Finally, several future studies could be pursued for WSNs. As described in chapter 13, the RPL protocol – despite being declared as *the* (only) routing protocol for WSNs – is not optimized for downward or P2P traffic. These traffic flows are of importance in networks with bidirectional traffic flows (*e.g.* in smart metering) and in networks with a significant amount of P2P traffic (*e.g.* home automation). A future study could investigate the overhead of RPL in these scenarios and compare RPL with alternative protocols, such for example reactive routing protocols.

Part VI

Appendices

Appendix A

List of Publications

A.1 Journal Publications

- U. Herberg, T. Clausen, “Study of Multipoint-to-Point and Broadcast Traffic Performance in the ’IPv6 Routing Protocol for Low Power and Lossy Networks’ (RPL)”, accepted for the Journal of Ambient Intelligence and Humanized Computing, Springer, 2011
- U. Herberg, T. Clausen, “Security Issues in the Optimized Link State Routing Protocol version 2”, International Journal of Network Security & Its Applications, special issue April, 2010

A.2 Conference and Workshop Publications

- T. Clausen, U. Herberg, “Some Considerations on Routing In Particular and Lossy Environments”, Proceedings of the 1st IAB Interconnecting Smart Objects with the Internet Workshop, March 2011
- U. Herberg, T. Clausen, “Yet Another Autoconf Proposal (YAAP) for Mobile Ad hoc NETworks”, Proceedings of the 6th International Conference on Mobile Ad-hoc and Sensor Networks (MSN), December 2010
- T. Clausen, U. Herberg, “Comparative Study of RPL-Enabled Optimized Broadcast in Wireless Sensor Networks”, Proceedings of the 6th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), December 2010
- U. Herberg, T. Clausen, R. Cole, “MANET Network Management and Performance Monitoring for NHDP and OLSRv2”, Proceedings of the 6th International Conference on Network and Services Management (CNSM), October 2010
- T. Clausen, U. Herberg, “Multipoint-to-Point and Broadcast in RPL”, Proceedings of the 1st International Symposium on Frontiers in Ubiquitous Computing, Networking and Applications (NeoFUSION), September 2010

- T. Clausen, U. Herberg, “Router and Link Admittance Control in the Optimized Link State Routing Protocol version 2 (OLSRv2)”, Proceedings of the 4th International Conference on Network and System Security (NSS), September 2010
- U. Herberg, T. Clausen, J. Milan, ”Digital Signatures for Admittance Control in the Optimized Link State Routing Protocol version 2”, Proceedings of the International Conference on Internet Technology and Applications (iTAP), August 2010
- T. Clausen, U. Herberg, “Vulnerability Analysis of the Optimized Link State Routing Protocol version 2 (OLSRv2)”, Proceedings of the International Conference on Wireless Communications, Networking and Information Security (WCNIS), June 2010
- U. Herberg, I. Taylor, “Development Framework for Supporting Java NS2 Routing Protocols”, Proceedings of the International Workshop on Future Engineering, Applications and Services (FEAS), May 2010
- U. Herberg, “Performance Evaluation of using a Dynamic Shortest Path Algorithm in OLSRv2”, Proceedings of the 8th Annual Conference on Communication Networks and Services Research (CNSR), May 2010
- E. Baccelli, T. Clausen, U. Herberg. C. Perkins, “IP Links in Multihop Ad Hoc Wireless Networks?”, Proceedings of the 18th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), September 2009
- U. Herberg, “JOLSRv2: An OLSRv2 implementation in Java”, Proceedings of the 4th OLSR Interop Workshop, 2008

A.3 Research Reports

- T. Clausen, U. Herberg, “Some Considerations on Routing In Particular and Lossy Environments”, INRIA research report 7540, February 2011
- U. Herberg, “Performance Analysis of SNMP in OLSRv2-routed MANETs, INRIA research report 7407, October 2010
- T. Clausen, U. Herberg, “Study of Multipoint-to-Point and Broadcast Traffic Performance in RPL”, INRIA research report 7384, September 2010
- T. Clausen, U. Herberg, “Yet Another Autoconf Proposal (YAAP) for Mobile Ad hoc NETworks”, INRIA research report 7341, July 2010
- T. Clausen, U. Herberg, “MANET Network Management and Performance Monitoring for NHDP and OLSRv2”, INRIA research report 7311, June 2010
- T. Clausen, U. Herberg, “Comparative Study of RPL-Enabled Optimized Broadcast in Wireless Sensor Networks”, INRIA research report 7296, May 2010

- T. Clausen, U. Herberg, "Router and Link Admittance Control in the Optimized Link State Routing Protocol version 2 (OLSRv2)", INRIA research report 7248, April 2010
- T. Clausen, U. Herberg, "Multipoint-to-Point and Broadcast in RPL", INRIA research report 7244, April 2010
- T. Clausen, U. Herberg, "Security Issues in the Optimized Link State Routing Protocol version 2 (OLSRv2)", INRIA research report 7218, March 2010
- T. Clausen, U. Herberg, J. Milan, "Digital Signatures for Admittance Control in the Optimized Link State Routing Protocol version 2", INRIA research report 7216, February 2010
- T. Clausen, U. Herberg, "Vulnerability Analysis of the Optimized Link State Routing Protocol version 2 (OLSRv2)", INRIA research report 7203, February 2010
- U. Herberg, N. Mariyasagayam, T. Clausen, "Comparison of NHDP and MHVB for Neighbor Discovery in Multi-hop Ad Hoc Networks", INRIA research report 7173, January 2010
- U. Herberg, "Performance Evaluation of using a Dynamic Shortest Path Algorithm in OLSRv2", INRIA research report 7174, January 2010
- U. Herberg, "Integrating Java Support for Routing Protocols in Ns2", INRIA research report 7075, October 2009
- T. Clausen, U. Herberg, "AUTOCONF - Stating the Problem", INRIA research report 6376, December 2007

A.4 Internet Drafts

Before a document is published as "Request for Comment (RFC)" by the Internet Engineering Taskforce (IETF), it has to run through a standardization process, typically starting as individual draft, which can be submitted by everyone without peer review. As such, it is available for discussion by other interested participants on the mailing list, but simply expresses an individual proposal of a person. If a Working Group is interested in the work, it can agree hat this draft becomes a Working Group draft. Once the Working Group agrees that after a certain number of revisions of the draft, it is mature for publication, the draft is sent to the Internet Engineering Steering Group (IESG) for consideration to publish it as RFC.

Following is a list of all my published Working Group drafts and individual drafts, including the history of all revisions.

A.4.1 Working Group Internet Drafts

- U. Herberg, R. Cole, I. Chakeres, "Definition of Managed Objects for the Neighborhood Discovery Protocol", Internet Draft (work in progress), draft-ietf-manet-nhdp-mib-07, January 2011

- U. Herberg, R. Cole, T. Clausen, “Definition of Managed Objects for the MANET Optimized Link State Routing Protocol version 2”, Internet Draft (work in progress), draft-ietf-manet-olsrv2-mib-03, January 2011
- U. Herberg, R. Cole, I. Chakeres, “Definition of Managed Objects for the Neighborhood Discovery Protocol”, Internet Draft (work in progress), draft-ietf-manet-nhdp-mib-06, November 2010
- U. Herberg, R. Cole, I. Chakeres, “Definition of Managed Objects for the Neighborhood Discovery Protocol”, Internet Draft (work in progress), draft-ietf-manet-nhdp-mib-05, November 2010
- U. Herberg, T. Clausen, “MANET Cryptographical Signature TLV Definition”, Internet Draft (work in progress), draft-ietf-manet-packetbb-sec-02, November 2010
- U. Herberg, R. Cole, I. Chakeres, “Definition of Managed Objects for the Neighborhood Discovery Protocol”, Internet Draft (work in progress), draft-ietf-manet-nhdp-mib-04, July 2010
- U. Herberg, R. Cole, T. Clausen, “Definition of Managed Objects for the MANET Optimized Link State Routing Protocol version 2”, Internet Draft (work in progress), draft-ietf-manet-olsrv2-mib-02, July 2010
- U. Herberg, T. Clausen, “MANET Cryptographical Signature TLV Definition”, Internet Draft (work in progress), draft-ietf-manet-packetbb-sec-01, July 2010
- U. Herberg, T. Clausen, “MANET Cryptographical Signature TLV Definition”, Internet Draft (work in progress), draft-ietf-manet-packetbb-sec-00, June 2010
- U. Herberg, R. Cole, I. Chakeres, “Definition of Managed Objects for the Neighborhood Discovery Protocol”, Internet Draft (work in progress), draft-ietf-manet-nhdp-mib-03, March 2010
- U. Herberg, R. Cole, I. Chakeres, “Definition of Managed Objects for the Neighborhood Discovery Protocol”, Internet Draft (work in progress), draft-ietf-manet-nhdp-mib-02, November 2009
- U. Herberg, R. Cole, T. Clausen, “Definition of Managed Objects for the MANET Optimized Link State Routing Protocol version 2”, Internet Draft (work in progress), draft-ietf-manet-olsrv2-mib-01, November 2009
- U. Herberg, R. Cole, I. Chakeres, “Definition of Managed Objects for the Neighborhood Discovery Protocol”, Internet Draft (work in progress), draft-ietf-manet-nhdp-mib-01, October 2009

A.4.2 Individual Internet Drafts

- U. Herberg, T. Clausen, “MANET Cryptographical Signature TLV Definition”, Internet Draft (work in progress), draft-herberg-manet-packetbb-sec-03, March 2010
- U. Herberg, T. Clausen, “Security Threats for NHDP”, Internet Draft (work in progress), draft-herberg-manet-nhdp-sec-threats-00, November 2009
- U. Herberg, T. Clausen, “Cryptographical Signatures in NHDP”, Internet Draft (work in progress), draft-herberg-manet-nhdp-sec-00, November 2009
- U. Herberg, T. Clausen, “MANET Cryptographical Signature TLV Definition”, Internet Draft (work in progress), draft-herberg-manet-packetbb-sec-02, July 2009
- U. Herberg, T. Clausen, “MANET Cryptographical Signature TLV Definition”, Internet Draft (work in progress), draft-herberg-manet-packetbb-sec-01, July 2009
- U. Herberg, T. Clausen, “MANET Cryptographical Signature TLV Definition”, Internet Draft (work in progress), draft-herberg-manet-packetbb-sec-00, July 2009

Appendix B

Usage Instructions of AgentJ

This chapter describes how to use a Java routing protocol implementation with NS2 by using AgentJ. It details all necessary steps for integrating the implementation, plus some optional, more advanced features. The background and the architecture of routing functionalities in AgentJ are detailed in chapter 3.

B.1 Basic Integration

This section describes the integration of a routing protocol written in Java into NS2 using the AgentJ library as extended as described in this document. It assumes that AgentJ is properly installed, as described in the AgentJ manual [76].

Note that for C++ implementations, each new protocol implementation requires NS2 modifications, and a subsequent recompilation of NS2. However, once AgentJ has been installed, no recompilation of NS2 is necessary any more, even when a new routing protocol implementation in Java is created and added. In the following step-by-step instruction, a demonstrative example routing protocol called `MyRoutingProtocol` is used to show the interaction between the protocol implementation, AgentJ, and NS2.

B.1.1 Addition of a Helper Class as Single Point of Entry for NS2

As the single point of entry for NS2 to the routing protocol implementation, a new helper class in Java needs to be created. This helper class – as depicted on the right side of figure B.1 – serves the same purpose as the `main` method of Java applications: to allow NS2 / AgentJ to "execute" the routing protocol implementation. In addition, the helper class provides an interface with two methods, called by NS2 for the purpose of routing. All other calls, such as for sending and receiving control packets or setting timers, are directly hooked into NS2 (by virtue of bytecode rewriting) and do not need any changes in the Java routing protocol implementation or in the helper class.

The Java helper class (in the following example called `MyRoutingAgent`) needs to extend

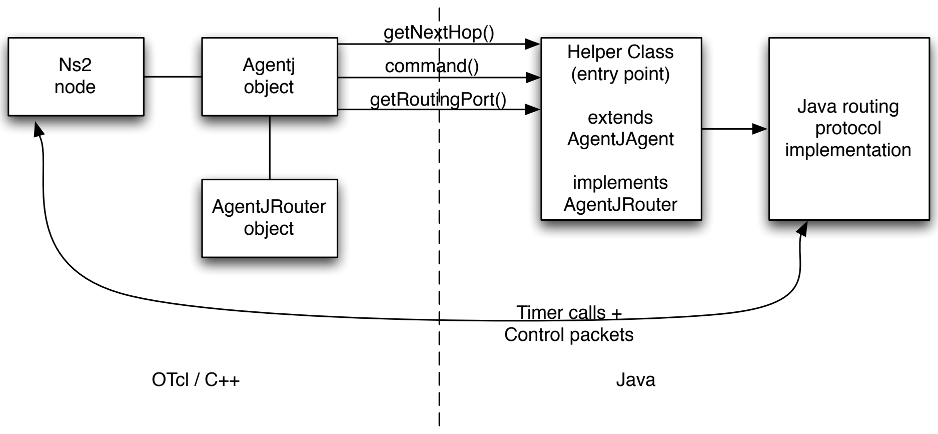


Figure B.1: The helper class on the right side serves as single point of entry from NS2 to the Java routing protocol implementation.

from `agentj.AgentJAgent`, and also has to implement the `agentj.AgentJRouter` interface. An example class definition looks as the following:

```
public class MyRoutingAgent extends AgentJAgent implements AgentJRouter
```

By implementing the `AgentJRouter` interface, the three methods `getRoutingPort()`, `getNextHop(int)`, and `command(String, String[])`, must also be provided by `MyRoutingAgent`:

- `public int getRoutingPort()`

  This method must be implemented such that it returns the UDP port number that the Java routing agent is running on. This can be any arbitrary number between 0 and 65535. NS2 invokes this method at the beginning of the simulation, in order to acquire the UDP port number; subsequently, any traffic received on that port is considered as control traffic and handed off to the Java routing protocol, which is listening on a "DatagramSocket".

- `public int getNextHop(int destination)`

  This method is called from NS2 whenever a unicast data packet arrives at the node. The Java method should return the next hop for the given destination (as NS2 node ID[1]) or -1 if no such destination has been found in the routing table. If the Java routing protocol implementation uses real IP Addresses (*i.e.* "java.net.InetAddress") and not only NS2 node IDs, this method must perform a mapping between these two address types.

  An exemplary mapping (limited to node IDs smaller than 256) would be:
  `InetAddress.getByName("0.0.0." + destination)`

  In a future version of AgentJ, the mapping could be done within the AgentJ code.

[1]In NS2, every node has a unique identifier, the NS2 node ID. Per default, this ID is an unsigned integer (including 0).

- `public boolean command(String command, String[] args)`

  The method `command(...)` evaluates the given command and its arguments. The method can be invoked at any time from the simulation script, notably at the beginning of the simulation to start the Java routing protocol (*i.e.* serves as the single point of entry). In the before-mentioned example, calling the command `startRouting` would start the Java routing protocol. Optionally, any other command can be added in the `command()` method, such as for outputting the routing table as in the following example:

```
public boolean command(String command, String[] args) {
   if (command.equals("startRouting")) {
      // code needs to be added here to start the routing protocol
      return true;
   } else if (command.equals("print_rtable")) {
      // code needs to be added here to output routing table
      return true;

   return false;
}
```

B.1.2 Installation of the Java Classes

In order to run the Java routing protocol, AgentJ has to find the corresponding class files. The location of the Java classes of the routing protocol must be added to an environmental variable called `AGENTJ_CLASSPATH`. In Linux, this is done by running:

```
export AGENTJ_CLASSPATH=.:/path/to/classfiles
```

where `/path/to/classfiles` needs to be replaced by the location of the Java classes. This line should be added to the `$HOME/.bashrc` file (if using the bourne shell) in order to execute it every time the user logs on.

B.1.3 Scenario Tcl Script

The scenario Tcl file that is called by NS2 must tell AgentJ which Java routing protocol to use. A complete example Tcl script is included in the current AgentJ distribution. Note in particular the following lines, specific to using a routing protocol with AgentJ, that represent a minimum set of parameters that have to be defined in the Tcl script:

1. **Define the routing agent:**

```
set opt(rp)        AgentJ     ;# Routing Protocol
...
$ns_ node-config \
                 -adhocRouting $opt(rp) \
...
```

The routing agent is set to "AgentJ" whatever Java routing protocol is used. This avoids adding a new routing protocol in `ns-2.34/tcl/lib/ns-lib.tcl` for every new Java routing protocol implementation.

2. **Attach the Java agent to a node**

```
set node [ $ns_ node ] ;# creates a new node

# The following line must be changed to reflect
# the name of the Java class.
[$node set ragent_] attach-agentj my.personal.MyRoutingAgent

# The following lines needs only be changed for advanced settings
# (refer to section 4).
[$node set ragent_] agentj setRouterAgent Agent/AgentJRouter
$ns_ at 0.0 "[$node set ragent_] agentj startRouting"
```

The parameter `my.personal.MyRoutingAgent` must be changed to the complete name of the Java class name of the helper class that has been added (as described in section B.1.1).

B.1.4 Change Addressing Scheme

The `$AGENTJ/conf/agentj.properties` file has to be modified as follows:

```
#If you want to change to IPv4 or IPv6 addressing,
#you have to add a line:

#java.net.preferIPv4Stack=true #or
#java.net.preferIPv6Stack=true
```

If the Java routing protocol uses only multicast addressing and no unicast addresses, nothing needs to be changed. In particular, in order to run the AgentJ examples in `$AGENTJ/examples/udp`, the file should not be modified. However, in order to use unicast addresses in the routing protocol, the line that corresponds to the preferred address family (IPv4 or IPv6) needs to be uncommented.

Note that these modifications are performed to the config files, read at execution time, and so do not require recompilation of AgentJ nor NS2 nor the routing protocol implementation.

B.1.5 Running the NS2 Simulation

The NS2 simulation can be started by launching the Tcl file

```
ns my-sample-script.tcl
```

The most common error at this point is, that a `java.lang.ClassNotFoundException` is thrown. In that case, it has to be assured that the location of the Java classes of the routing protocol have been correctly included in the `AGENTJ_CLASSPATH` environmental variable, as described in section B.1.2.

B.2 Advanced Features

Section B.1 described the basic and necessary steps to interface a Java routing protocol in AgentJ. This section presents some more advanced features of using Java routing protocols in AgentJ.

Section B.2.1 explains the steps to include more detailed tracing of control packets. Using this modification allows to output details about the payload of a control packet in NS2 trace files.

Section B.2.2 details how to change the specific behavior of the routing protocol when a data packet arrives at an NS2 node. The default behavior of looking up unicast addresses for the destination of a data packet can be modified. For example, for a reactive routing protocol, a route discovery phase could be initiated.

B.2.1 Tracing of Routing Control Packets

Every control packet that is sent from the Java routing agent will appear in the NS2 trace file[2], just as any control packet sent by a native C++ implementation of a routing protocol. Per default, all control traffic packets sent by the Java agent are marked as `AGENTJ` packets. The following NS2 trace file line represent a typical exemplary output (note that the line-breaks in the following trace entry do not occur in the NS2 trace file).

```
s -t 0.004000000 -Hs 39 -Hd -2 -Ni 39 -Nx 1216.36 -Ny 594.63 -Nz 0.00 -Ne -1.0
-Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 39.9999 -Id -1.50000 -It AGENTJ
-Il 25 -If 0 -Ii 0 -Iv 254
```

Essentially, the line means that a packet has been sent (first letter '`s`') from RTR layer (`-Nl RTR`) at time 0.004 seconds after simulations start (`-t 0.004`). The packet originates from the node with the ID 39 (`-Hs 39`), is destined to port 50000 using the broadcast address (`-Id -1.50000`), and is of type AGENTJ (`-It AGENTJ`). For a detailed description of the trace file format, refer to the NS2 manual [59].

Sometimes, it is desirable to display information from the payload of such a packet. This section describes how to add more detailed packet traces. Note that this advanced modification requires a recompilation of NS2.

[2]Refer to the NS2 manual [59] for more details about tracing.

Step 1:

If desired, the displayed name of a control packet in the trace file can be changed from the default value `AGENTJ` to any other name. The file `ns-2.34/common/packet.h` has to be changed as described in the following:

```
name_[PT_DSR]= "DSR";
name_[PT_AODV]= "AODV";

// change "AGENTJ" to the desired name
name_[PT_AGENTJ] = "AGENTJ";
```

Step 2:

For adding a detailed trace of the control traffic packet, the file `NS2.34/trace/cmu-trace.cc` has to be modified:

```
void CMUTrace::format(Packet* p, const char *why)
{
        hdr_cmn *ch = HDR_CMN(p);
        int offset = 0;
        ...
                switch(ch->ptype()) {
                case PT_AODV:
                        format_aodv(p, offset);
                        break;
                case PT_TORA:
                        format_tora(p, offset);
                        break;
```

Specifically, a call to a format method has to be added:

```
case PT_AGENTJ:
        format_myrouting(p, offset);
        break;
```

Next, a method `format_myrouting(Packet*, int)` has to be included in `cmu-trace.cc` that formats the output string in the trace file for the control packet. This method will be similar to the other format methods such as `format_aodv` or `format_tora`. For more information about writing a CMU trace format method, refer to the NS2 manual [59].

In principle, it would also be possible to write a callback handler to the Java routing protocol implementation, which returns a string that will be output to the trace file. This is not part of the current version of AgentJ, but could be added at a later point of time.

NS2 has to be recompiled after these modifications (*i.e.* `configure` and `make`).

The following NS2 trace file line represent a typical exemplary output after the beforementioned modifications:

```
s -t 0.004000000 -Hs 39 -Hd -2 -Ni 39 -Nx 1216.36 -Ny 594.63 -Nz 0.00 -Ne -1.0
-Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 39.9999 -Id -1.50000 -It AGENTJ -Il
25 -If 0 -Ii 0 -Iv 254 -P olsrv2 -Pn 1 -Ps 7996 [-Pt HELLO -Po 1 -Ph 1 -Pms 4104]
```

All parameters in that line that have a prefix "`-P`", are created by the formatting function that has been defined by the user.

B.2.2 Changing the Behavior of the Router when Data Packets arrive

AgentJ provides a certain default behavior whenever a packet arrives at the routing layer. If it is an incoming packet from the MAC layer, AgentJ will check whether it is a control packet or a data packet. Control packets will be forwarded to the listening socket in the Java routing protocol. This socket is the one that is bound to the port number given by the method `getRoutingPort()` in the helper class. Data packets are delivered to the application layer of the NS2 node. Outgoing packets will be separated into unicast and broadcast traffic. For unicast packets, AgentJ calls `getNextHop()` of the Java routing protocol to get a valid next hop for the given destination. Section 3.4 gives more details on this.

This particular behavior is implemented in the method `receivePacket()` of the C++ file `$AGENTJ/core/src/main/c/agentj/AgentJRouter.cpp`. This class can be subclassed to change the above-mentioned default behavior for incoming and outgoing packets.

The necessary steps to subclass the `AgentJRouter` class are presented in the following. In this example, the subclass will be called `MyRouter`.

Step 1:

The new files `MyRouter.h` and `MyRouter.cpp` have to be created in the directory `$AGENTJ/core/src/main/c/agentj`. These files should at least contain the following fields and methods:

- MyRouter.h:

```
#ifndef _MYROUTER
#define _MYROUTER
#include "AgentJRouter.h"

class MyRouter : public AgentJRouter {
    public:
        MyRouter() : AgentJRouter(){}
        ~MyRouter(){}

        void receivePacket(Packet *p,Handler *handle);
        bool ProcessCommands(int argc, const char* const* argv);
};

#endif // _MYROUTER
```

- MyRouter.cpp:

```
#include "MyRouter.h"

static class MyRouterInstantiator : public TclClass {
    public:
        MyRouterInstantiator() :
            TclClass("Agent/AgentJRouter/MyRouter") {}
        TclObject *create(int argc, const char*const* argv) {
            return (new MyRouter());
        }
} my_router;


void MyRouter::receivePacket(Packet *p,Handler *handle) {
    // treat an incoming packet
}
```

Step 2:
The new class has to be added to the `Makefile.in` of NS2. The rule can be added to the `OBJ_AGENTJ_CPP` makefile variable:

```
OBJ_AGENTJ_CPP = $(AGENTJ_UTILS)/LinkedList.o \
[...]
      $(JAVM)/TimerWrapper.o $(JAVM)/JAVMTimer.o \
      $(AGENTJ_C_SRC)/MyRouter.o
```

NS2 has to be recompiled after these modifications (*i.e.* `configure` and `make`).

Step 3: In the scenario Tcl file, AgentJ has to be instructed to use the subclass instead of the default `AgentJRouter`. At the point in the Tcl file where the nodes are created and the Java agent is attached, the following line must be changed from:

```
[$node_($i) set ragent_] agentj \
        setRouterAgent Agent/AgentJRouter
```

to:

```
[$node_($i) set ragent_] agentj \
        setRouterAgent Agent/AgentJRouter/MyRouter
```

B.2.3 Limitations of AgentJ with Routing Functionalities

As of early 2011, AgentJ only supports routing protocols which send control traffic over UDP. TCP protocols (such as BGP) are not currently supported. In addition, it is not possible to run a simulation with nodes using IPv4 and IPv6 addresses at the same time. All nodes have to use the same address family.

Appendix C

Usage Instructions of JOLSRv2

This chapter details how to install and to launch JOLSRv2, which has been presented in chapter 7.

C.1 Installation Instructions

The following steps have to be performed to install JOLSRv2:

- The archive has to be downloaded and extracted in any folder
- If more than one MANET interface is to be used, the UDPSocket library needs to be used, which is included in the archive. For Linux and Mac OS, the location of the UDPSocket library has to be added to the `LD_LIBRARY_PATH` (*e.g.* the line

  ```
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/udpsocketlibrary
  ```

  has to be added to the file $HOME/.bashrc when using the Bash shell). Currently, Windows does not support multiple interfaces in JOLSRv2.
- The `JOLSRV2_RT` library has to be added to the `LD_LIBRARY_PATH`. This library is responsible for modifying the routes in the operating system routing table. More details are described in section 7.5.
- The configuration file "config.txt" has to be modified. Example:

  ```
  [default]
  ipversion = 4
  interfaces = wlan0

  [wlan0]
  HELLO_INTERVAL = 2000
  ```

For a complete list of the config file options, refer to section C.2.

- JOLSRv2 has to be launched with administrative user rights (*e.g.* using "sudo") with: ./jolsrv2.sh

C.2 Options in the Configuration File

The config.txt file is divided into one or several sections. Every section is headed by the section name in brackets (e.g. [default]). There has to be a section [default] with the possible contents detailed in table C.1.

Table C.1: JOLSRv2 router parameters in the config file

Entry	**Description**
ipversion = version	version may be 4 or 6 (default: 4)
interfaces = ifaceList	ifaceList is a space-separated list of interfaces (*e.g.* eth0 wlan0)
rmi = boolean_value	whether to use the RMI remote access (default: false)
debug = boolean_value	whether to output detailed debugging information (default: false)
dumping = boolean_value	when used in NS2, whether to output topology information in the trace file (for details refer to section 2.6.4) (default: false)
triggerTC = boolean_value	whether this router will trigger TC messages (default: true)
routerclass = classname	name of a Java class that is used instead of the default OLSRv2Router class (refer to section 7.6 for more details)
`I_HOLD_TIME` = time	time in milliseconds
`N_HOLD_TIME` = time	
`A_HOLD_TIME` = time	
`F_HOLD_TIME` = time	
`O_HOLD_TIME` = time	
`P_HOLD_TIME` = time	
`T_HOLD_TIME` = time	
`TC_HOP_LIMIT` = time	
`I_HOLD_TIME` = time	
`TC_INTERVAL` = time	
`TC_MIN_INTERVAL` = time	
`TP_MAXJITTER` = time	
`TT_MAXJITTER` = time	
Willingness = value	value must be between `WILL_NEVER` and `WILL_ALWAYS`

In addition to the router parameters, any interface parameter may be specified and is then applied to the specified interface. Refer to table C.2 for a list of interface parameters. There must be a section for every interface listed in the value `ifaceList`.

Table C.2: JOLSRv2 interface parameters in the config file

Entry	**Description**
ipaddresses = {address}*	address is a valid IP address for that interface
broadcast = broadcast_address	a broadcast address (*e.g.* 255.255.255.255)
manet = isManet	determines whether this interface is a MANET interface (true or false, default is true)
interfaceclass = classname	name of a Java class that is used instead of the default OLSRv2Interface class (refer to section 7.6 for more details)
use_link_quality = boolean_value	whether to use link quality based on packet sequence number (default: false)
use_L2_link_quality = boolean_value	whether to use link quality based on missing ACKs on L2 (default: false)
triggerHELLO = boolean_value	whether this interface will trigger HELLO messages (default: true)
sender_HELLO = boolean_value	whether this interface sends any HELLO messages (default: true)
sender_TC = boolean_value	whether this interface sends any TC messages (default: true)
`HELLO_INTERVAL` = interval	in milliseconds
`HELLO_MIN_INTERVAL` = interval	
`HYST_ACCEPT` = time	
`HYST_REJECT` = time	
`H_HOLD_TIME` = time	
`L_HOLD_TIME` = time	
`HP_MAXJITTER` = time	
`HT_MAXJITTER` = time	
`RX_HOLD_TIME` = time	

C.3 Running JOLSRv2 in NS2

This section describes how to instantiate JOLSRv2 in the network simulator NS2, based on the general description of appendix B.

C.3.1 JOLSRv2 Helper Class

JOLSRv2 provides a "helper class", as described in section B.1.1, denoted `net.jolsrv2.simulation.ns2.Ns2jOLSRv2Agent`. In particular, this helper class implements the method `command()`, which evaluates the given command and its arguments and can be invoked at any time from the simulation script, notably at the beginning of the simulation to start JOLSRv2. The implemented method accepts the commands listed in table C.3.

Table C.3: Accepted arguments for the `command()` method of the JOLSRv2 helper class

Command	**Description**
`startProtocol`	Starts JOLSRv2.
`print_rtable`	Prints the content of the routing set to the logger (refer to section 7.4 for a description of logging in JOLSRv2).
`stop`	Stops the protocol (*i.e.* no more control messages are sent, and all information bases are cleared).
`restart`	Restarts JOLSRv2 if it has been stopped before.
`trigger-hello`	Triggers a HELLO.

C.3.2 Custom JOLSRv2 Trace File Logging

Based on the description of section B.2.1, a custom trace file formatting for OLSRv2 messages has been added to NS2 logging, allowing to log details about the content of HELLO and TC messages. An example log entry is:

```
s -t 499.284900466 -Hs 25 -Hd -2 -Ni 25 -Nx 539.66 -Ny 404.97 -Nz 0.00 -Ne
100.000000 -Nl MAC -Nw --- -Ma 0 -Md ffffffff -Ms 19 -Mt 800 -Is 25.9999 -Id
-1.50000 -It OLSRV2 -Il 133 -If 0 -Ii 39987 -Iv 254 -P olsrv2 -Pn 2 -Ps 10707
[-Pt TC -Po 33 -Ph 254 -Pms 19589 -Pa 57;25;51;12]
[-Pt TC -Po 12 -Ph 253 -Pms 5910 -Pa ]
```

The protocol-specific logging for OLSRv2 starts with the parameter `-P olsrv2`, followed by `Pn` for the number of messages in the packet, `Ps` for the sequence number of the packet, and then in brackets information about each message in the packet. The following message parameters are defined: `Pt` for the message type (HELLO or TC), `Po` for the ID of the node originating the message, `Ph` for the TTL of the message, `Pms` for the message sequence number, and `Pa` for a list of node IDs that are advertised in the message, separated by semicolons.

C.3.3 JOLSRv2 Router Agent

Based on section B.2.2, a specific "router agent" has been developed for JOLSRv2, denoted `OLSRv2Router`. The agent provides the same functionality as the standard `AgentJRouter` agent, described in section B.2.2. However, it marks control messages with the packet type `PT_OLSRV2`, allowing the OLSRv2-specific logging of control messages, described in section C.3.2.

The router agent for JOLSRv2 must be set as described in section B.2.2 with

```
[$node_($i) set ragent_] agentj \
        setRouterAgent Agent/AgentJRouter/OLSRv2Router
```

Another router agent for JOLSRv2, called `OLSRv2LinkbufferRouter`, is provided, which implements the link buffering behavior as presented in chapter 10.

Appendix D

Usage Instructions of JRPL

This chapter details how to install and to launch JRPL, which has been presented in chapter 14.

D.1 Installation Instructions

The following steps have to be performed to install JRPL:

- The archive has to be downloaded and extracted in any folder
- The configuration file “config.txt” has to be modified. Example:

  ```
  [default]
  root = false
  debug = true
  dumping = false
  trickle=true
  ```

  For a complete description of the config file options, refer to section D.2.

- JRPL is launched with:
  java net.rpl.RPLRouter

D.2 Options in the Configuration File

The config.txt file is divided into one or several sections. Every section is headed by the section name in brackets (e.g. [default]). There has to be a section [default] with the possible contents detailed in table D.1.

Table D.1: JRPL parameters in the config file

Entry	Description
root = boolean_value	if set to `true`, this router is the root (is ignored when JRPL is executed in NS2; default is false)
debug = boolean_value	whether to output detailed debugging information (default is false)
dumping = boolean_value	when used in NS2, whether to output topology information in the trace file (for details refer to section 2.6.4; default value is false)
trickle = boolean_value	if set to `true`, the trickle timer is used, otherwise a periodic timer is used (default is false)
DAO = boolean_value	if set to `true`, the downward flow mechanism is activated (*i.e.* DAOs are sent; default value is false)
DAOACK = boolean_value	if set to `true` and if "DAO" is also set to `true`, DAOs are acknowledged with DAOACKs by the root (default value is false)
TRAFFIC_JITTER = time	time in milliseconds to jitter data traffic (default is 0, *i.e.* jitter is disabled)
NUD = value	value can either be "" (empty string), "NHDP" or "God", depending on which NUD mechanism is used (default is "")

D.3 Running JRPL in NS2

This section explains how to instantiate JRPL in the network simulator NS2, based on the general description of appendix B.

D.3.1 JRPL Helper Class

JRPL provides a "helper class", as described in section B.1.1, denoted `net.rpl.ns2.Ns2RPLAgent`. In particular, this helper class implements the method `command()`, which evaluates the given command and its arguments and can be invoked at any time from the simulation script, notably at the beginning of the simulation to start JRPL. The implemented method accepts the commands (together with their command arguments) listed in table D.2.

D.3.2 Custom JRPL Trace File Logging

Based on the description of section B.2.1, a custom trace file formatting for RPL messages has been added to NS2 logging, allowing to log RPL messages, similar to the logging presented in section C.3.2.

The protocol-specific logging for JRPL defines the following parameters: `Pt` for the message type (DIO, DAO, DAOACK, DOWNWARDTRAFFIC, UPWARDTRAFFIC, FLOODING-TRAFFIC and HELLO, the latter for the NHDP NUD extension described in section 14.6), `Po` for the node ID of the message originator, `Pd` for the ID of the node to which the message is destined, `Ph` for the TTL of the message, and `Pms` for the message sequence number.

Table D.2: Accepted arguments for the `command()` method of the JRPL helper class

Command	Description
startProtocol	Starts JRPL. The NS2 node with the ID of 0 is the root.
start_traffic [packetsize] [interval]	Starts sending flooding traffic, as presented in chapter 16. This can only be called on the root. [packetsize] defines the size of each data packet in octets. [interval] defines the transmission interval in seconds.
stop_traffic	Stops sending flooding traffic. This can only be called on the root.
start_downward_traffic [packetsize] [interval] [destination]	Starts sending downward traffic. This can only be called on the root. [packetsize] defines the size of each data packet in octets. [interval] defines the transmission interval in seconds. [destination] is the ID of the NS2 node to which the traffic packets are sent.
stop_downward_traffic [destination]	Stops sending downward traffic. This can only be called on the root. [destination] is the ID of the NS2 node to which the traffic packets were sent.
start_upward_traffic [packetsize] [interval]	Starts sending upward traffic. This cannot be called on the root. [packetsize] defines the size of each data packet in octets. [interval] defines the transmission interval in seconds.
stop_upward_traffic	Stops sending upward traffic. This cannot be called on the root.

D.3.3 JRPL Router Agent

Based on section B.2.2, a specific "router agent" has been developed for JRPL, denoted `RPLRouter`. The agent provides the same functionality as the standard `AgentJRouter` agent, as described in section B.2.2. However, it marks control messages with the packet type `PT_RPL`, allowing the RPL-specific logging of control messages, as described in section D.3.2.

The router agent for JRPL must be set as described in section B.2.2 with

```
[$node_($i) set ragent_] agentj \
        setRouterAgent Agent/AgentJRouter/RPLRouter
```

List of Figures

List of Tables

Bibliography

[1] I. S. Consortium, “ISC Domain Survey,” http://www.isc.org/solutions/survey.

[2] “AS65000 BGP Table Statistics,” http://bgp.potaroo.net/as2.0/bgp-active.html.

[3] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, “BGP routing stability of popular destinations,” in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, 2002.

[4] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, “BGP routing dynamics revisited,” *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 5–16, March 2007.

[5] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, “Delayed Internet routing convergence,” *Networking, IEEE/ACM Transactions on*, vol. 9, no. 3, pp. 293 –306, Jun. 2001.

[6] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, “Observation and analysis of BGP behavior under stress,” in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, 2002.

[7] F. Baker, “An outsider’s view of MANET,” March 2002, Internet Draft, work in progress, draft-baker-manet-review-01.txt.

[8] A. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall Professional Technical Reference, 2002.

[9] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, and J. Postel, “Internet Registry IP Allocation Guidelines,” November 1996, Best Current Practice 12, RFC 2050.

[10] S. Corson and J. Macker, “Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations,” January 1999, Informational RFC 2501.

[11] “Funkfeuer website,” http://www.funkfeuer.at.

[12] “Freifunk website,” http://www.freifunk.net.

[13] “Athens wireless metropolitan network (awmn) website,” https://awmn.net.

[14] “Seattlewireless website,” http://www.seattlewireless.net.

[15] H. Suzuki, Y. Kaneko, K. Mase, S. Yamazaki, and H. Makino, "An Ad Hoc Network in the Sky, SKYMESH, for Large-Scale Disaster Recovery," in *Proceedings of the 64th IEEE Vehicular Technology Conference*, Sept. 2006.

[16] "SMAVNET webpage," http://lis.epfl.ch/?content=research/projects/SwarmingMAVs.

[17] "BEAR webpage," http://robotics.eecs.berkeley.edu/bear.

[18] "RECUV webpage," http://recuv.colorado.edu.

[19] Scatterweb, "MSB430 Specification," http://www.scatterweb.com/content/downloads/datasheets/fact-sheet-msb430-v1.0-en.pdf.

[20] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 102 – 114, Aug. 2002.

[21] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," September 2007, Standards Track RFC 4862.

[22] R. Droms, "Dynamic Host Configuration Protocol," March 1997, Standards Track RFC 2131.

[23] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," July 2003, Standards Track RFC 3315.

[24] U. Herberg and T. Clausen, "Yet Another Autoconf Proposal (YAAP) for Mobile Ad hoc NETworks," in *Proceedings of the 6th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, December 2010.

[25] J. Moy, "OSPF Version 2," April 1998, Standards Track RFC 2328.

[26] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," January 2006, Standards Track RFC 4271.

[27] T. Henderson, P. Spagnolo, and J. Kim, "A wireless interface type for OSPF," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, Oct. 2003.

[28] C. Adjih, E. Baccelli, and P. Jacquet, "Link state routing in wireless ad-hoc networks," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, Oct. 2003.

[29] G. Tassey, "Standardization in technology-based markets," *Research Policy*, vol. 29, no. 4-5, pp. 587 – 602, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/B6V77-40379X9-9/2/28ef18d5d70cce39991cc68c855949cb

[30] IETF, "Web site," http://www.ietf.org.

[31] P. Hoffman and S. Harris, "The Tao of IETF: A Novice's Guide to the Internet Engineering Task Force," September 2006, Informational RFC 4677.

[32] H. Alvestrand, “A Mission Statement for the IETF,” October 2004, Best Current Practice 95, RFC 3935.

[33] IETF MANET working group, “Charter of the working group,” http://www.ietf.org/html.charters/manet-charter.html.

[34] T. Clausen, C. Dearlove, and B. Adamson, “Jitter Considerations in Mobile Ad Hoc Networks (MANETs),” February 2008, Standards Track RFC 5148.

[35] T. Clausen, C. Dearlove, J. Dean, and C. Adjih, “Generalized MANET Packet/Message Format,” February 2009, Standards Track RFC 5444.

[36] T. Clausen and C. Dearlove, “Representing Multi-Value Time in Mobile Ad Hoc Networks (MANETs),” March 2009, Standards Track RFC 5497.

[37] T. Clausen, C. Dearlove, and J. Dean, “Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP),” April 2011, Standards Track RFC 6130.

[38] I. Chakeres and C. Perkins, “Dynamic MANET On-demand (DYMO) Routing,” July 2010, Internet Draft, work in progress, draft-ietf-manet-dymo-21.txt.

[39] T. Clausen, C. Dearlove, and P. Jacquet, “The Optimized Link-state Routing Protocol version 2,” April 2010, Internet Draft, work in progress, draft-ietf-manet-olsrv2-11.txt.

[40] J. Macker, “Simplified Multicast Forwarding,” March 2011, Internet Draft, work in progress, draft-ietf-manet-smf-11.

[41] IETF AUTOCONF working group, “Charter of the working group,” http://www.ietf.org/html.charters/autoconf-charter.html.

[42] E. Baccelli and M. Townsley, “IP Addressing Model in Ad Hoc Networks,” September 2010, Informational RFC 5889.

[43] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur, “RPL: IPv6 Routing Protocol for Low power and Lossy Networks,” March 2011, Internet Draft, work in progress, draft-ietf-roll-rpl-19.

[44] U. Herberg and I. Taylor, “Development Framework for Supporting Java NS2 Routing Protocols,” in *Proceedings of the 2010 International Workshop on Future Engineering, Applications and Services (FEAS)*, May 2010.

[45] E. Baccelli, T. Clausen, U. Herberg, and C. Perkins, “IP Links in Multihop Ad Hoc Wireless Networks?” in *Proceedings of the 18th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, September 2009.

[46] U. Herberg, “JOLSRv2 – an OLSRv2 implementation in Java,” in *Proceedings of the 4th OLSR Interop workshop*, October 2008.

[47] ——, "Performance Evaluation of using a Dynamic Shortest Path Algorithm in OLSRv2," in *Proceedings of the 8th Annual Communication Networks and Services Research Conference (CNRS)*, 2010.

[48] U. Herberg, T. Clausen, and R. Cole, "MANET Network Management and Performance Monitoring for NHDP and OLSRv2," in *Proceedings of the 6th International Conference on Network and Services Management (CNSM)*, October 2010.

[49] U. Herberg, R. Cole, and I. Chakeres, "Definition of Managed Objects for the Neighborhood Discovery Protocol," January 2011, Internet Draft, work in progress, draft-ietf-manet-nhdp-mib-07.txt.

[50] U. Herberg, R. Cole, and T. Clausen, "Definition of Managed Objects for the Optimized Link State Routing Protocol version 2," January 2011, Internet Draft, work in progress, draft-ietf-manet-olsrv2-mib-03.txt.

[51] U. Herberg and T. Clausen, "Security Issues in the Optimized Link State Routing Protocol version 2 (OLSRv2)," *International Journal of Network Security & Its Applications (IJNSA)*, vol. 2, no. 2, 2010.

[52] T. Clausen and U. Herberg, "Vulnerability Analysis of the Optimized Link State Routing Protocol version 2 (OLSRv2)," in *Proceedings of the IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS)*, June 2010.

[53] U. Herberg, T. Clausen, and J. Milan, "Digital Signatures for Admittance Control in the Optimized Link State Routing Protocol version 2," in *Proceedings of the International Conference on Internet Technology and Applications (iTAP)*, August 2010.

[54] T. Clausen and U. Herberg, "Router and Link Admittance Control in the Optimized Link State Routing Protocol version 2 (OLSRv2)," in *Proceedings of the 4th International Conference on Network and System Security (NSS)*, September 2010.

[55] U. Herberg and T. Clausen, "MANET Cryptographical Signature TLV Definition," March 2011, Internet Draft, work in progress, draft-ietf-manet-packetbb-sec-03.txt.

[56] T. Clausen and U. Herberg, "Multipoint-to-Point and Broadcast in RPL," in *Proceedings of First International Symposium on Frontiers in Ubiquitous Computing, Networking and Applications (NeoFUSION)*, September 2010.

[57] ——, "Comparative Study of RPL-Enabled Optimized Broadcast in Wireless Sensor Networks," in *Proceedings of the 6th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, December 2010.

[58] U. Herberg and T. Clausen, "Study of Multipoint-to-Point and Broadcast Traffic Performance in the 'IPv6 Routing Protocol for Low Power and Lossy Networks' (RPL)," *Journal of Ambient Intelligence and Humanized Computing*, 2011.

[59] K. Fall and K. Varadhan, "The Network Simulator – NS-2," http://www.isi.edu/nsnam/ns.

[60] "QualNet website," http://www.scalable-networks.com/products/qualnet.

[61] "OMNeT website," http://www.omnetpp.org.

[62] "NS3 website," http://www.nsnam.org.

[63] E. Weingartner, H. vom Lehn, and K. Wehrle, "A Performance Comparison of Recent Network Simulators," in *Proceedings of the IEEE International Conference on Communications (ICC)*, June 2009.

[64] D. Wetherall, "Objective TCL," http://otcl-tclcl.sourceforge.net/otcl.

[65] Sourceforge, "GNUplot," http://www.gnuplot.info.

[66] "GNU screen webpage," http://www.gnu.org/software/screen.

[67] S. Kurkowski, T. Camp, N. Mushell, and M. Colagrosso, "A Visualization and Analysis Tool for NS-2 Wireless Simulations: iNSpect," in *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005.

[68] I. Taylor, B. Adamson, I. Downard, and J. Macker, "AgentJ: Enabling Java NS-2 Simulations for Large Scale Distributed Multimedia Applications," in *Proceedings of the 2nd International Conference on Distributed Frameworks for Multimedia Applications*, 2006.

[69] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET Simulation Studies: the Incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, 2005.

[70] Naval Research Laboratory, "Protean Protocol Prototyping Library (Protolib)," http://cs.itd.nrl.navy.mil/work/protolib.

[71] "Java Network Simulator," http://jns.sourceforge.net.

[72] "Javis: A Java NAM Visualization Tool," http://warriors.eecs.umich.edu/viz_tools/nam.html.

[73] "DRCL J-Sim," http://www.j-sim.org.

[74] "JiST: Java in Simulation Time Simulator," http://jist.ece.cornell.edu/index.html.

[75] S.Liang, *Java Native Interface: Programmer's Guide and Specification.* Prentice Hall PTR, 1999.

[76] Naval Research Lab, "AgentJ: Java Network Simulations in NS-2. An Installation and User Manual," http://cs.itd.nrl.navy.mil/work/agentj.

[77] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," February 2006, Standards Track RFC 4291.

[78] olsr.org, "Olsr.org OLSR implementation," http://olsr.org.

[79] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," October 2003, Experimental RFC 3626.

[80] T. Boot, "Analysis of MANET and NEMO," June 2007, Internet Draft, work in progress, draft-boot-manet-nemo-analysis-01.txt.

[81] K. Mase and K. Akima, "Prefix Distribution Framework for Connected MANETs," November 2007, Internet Draft, work in progress, draft-mase-autoconf-prefix-framework-00.txt.

[82] S. Ruffino, P. Stupar, T. Clausen, and S. Singh, "Connectivity Scenarios for MANET," January 2006, Internet Draft, work in progress, draft-ruffino-autoconf-conn-scenarios-00.txt.

[83] N. Shenoy, Y. Pan, and V. G. Reddy, "Bandwidth Reservation and QoS in Internet MANETs," in *Proceedings of the 14th ICCCN International Conference on Computer Communications and Networks*, October 2005.

[84] J. F. Kurose and K. W. Ross, *Computer Networking: a Top-Down Approach Featuring the Internet Package*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., 2000.

[85] O. Troan and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6," December 2003, Standards Track RFC 3633.

[86] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," September 2007, Standards Track RFC 4861.

[87] C. Bernardos, M. Calderon, and H. Moustafa, "Survey of IP address autoconfiguration mechanisms for MANETs," June 2010, Internet Draft, work in progress, draft-bernardos-manet-autoconf-survey-05.txt.

[88] M. Thoppian and R. Prakash, "A distributed protocol for dynamic address assignment in mobile ad hoc networks," *IEEE Transactions on Mobile Computing*, pp. 4–19, 2006.

[89] A. Tayal and L. Patnaik, "An address assignment for the automatic configuration of mobile ad hoc networks," *Personal and Ubiquitous Computing*, vol. 8, no. 1, pp. 47–54, 2004.

[90] M. Mohsin and R. Prakash, "IP address assignment in mobile ad hoc networks," in *Proceedings of the Military Communications Conference (MILCOM)*, 2002.

[91] J. Linton, "Automatic router configuration protocol," March 2002, Internet Draft, work in progress, draft-linton-arcp-00.txt.

[92] Y. Noisette and A. Williams, "A framework for zerouter operations," February 2003, Internet Draft, work in progress, draft-noisette-zerouter-frmwk-00.txt.

[93] A. White, "Zero-configuration subnet prefix allocation using UIAP," October 2002, Internet Draft, work in progress, draft-white-zeroconf-subnet-alloc-01.txt.

[94] K. Weniger, “PACMAN: Passive autoconfiguration for mobile ad hoc networks,” *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 3, pp. 507–519, 2005.

[95] I. Chakeres, “IANA Allocations for Mobile Ad Hoc Network (MANET) Protocols,” March 2009, Standards Track RFC 5498.

[96] P. Leach, M. Mealling, and R. Salz, “A Universally Unique IDentifier (UUID) URN Namespace,” July 2005, Standards Track RFC 4122.

[97] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing,” July 2003, Experimental RFC 3561.

[98] V. Fuller, T. Li, J. Yu, and K. Varadhan, “Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy,” September 1993, Standards Track RFC 1519.

[99] “UPPAAL website,” http://www.uppaal.org.

[100] B. Gebremichael, F. Vaandrager, and M. Zhang, “Analysis of the zeroconf protocol using UPPAAL,” in *Proceedings of the 6th International conference on Embedded software (EMSOFT)*, 2006.

[101] G. Behrmann, A. David, and K. G. Larsen, “A Tutorial on Uppaal,” http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf, November 2004.

[102] C. Dearlove, “New Capabilities in Security and QoS Using the Updated MANET Routing Protocol OLSRv2,” in *Proceedings of the Information Systems and Technology Panel (IST) Symposium*, December 2010.

[103] T. Cormen, C. Leiserson, R. R, and C. Stein, *Introduction to Algorithms.* MIT Press, 2001.

[104] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[105] T.Clausen, T.Olesen, and N.Larsen, “Investigating broadcast performance in Mobile Ad-hoc Networks,” in *Proceedings of the IEEE conference on Wireless Personal Multimedia Communications (WPMC)*, 2002.

[106] A. Qayyum, L. Viennot, and A. Laouiti, “Multipoint relaying: An efficient technique for flooding in mobile wireless networks,” in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS)*, 2001.

[107] “Java Logging Technology,” http://java.sun.com/javase/6/docs/technotes/guides/logging/index.html.

[108] Hipercom@LIX, “OLSR Interoperability Workshop,” http://interop.thomasclausen.org.

[109] “GNU diff website,” http://www.gnu.org/software/diffutils.

[110] "JOLSRv2 applets webpage," http://herberg.name/projects/79-olsrv2-implementation/72-interoperability-tests-article.

[111] D. Eppstein and J. Wang, "A steady state model for graph power laws," *Arxiv preprint cs/0204001*, 2002.

[112] P. Erdos and A. Renyi, "On random graphs," *Publ. Math. Debrecen*, vol. 6, no. 290-297, p. 156, 1959.

[113] olsr.org, "RFC 5444 implementation," http://olsr.org/git/?p=packetbb.git;a=summary.

[114] NRL, "NHDP implementation," http://cs.itd.nrl.navy.mil/work/nhdp/index.php.

[115] CRC, "OLSRv2 implementation," http://www.crc.gc.ca/en/html/crc/home/mediazone/eye_on_tech/2009/issue10/olsr.

[116] Niigata University, "RFC 5444 implementation," http://www2.net.ie.niigata-u.ac.jp/nOLSRv2/olsrv2/Welcome.html.

[117] NRL, "CORE network emulator," http://cs.itd.nrl.navy.mil/work/core.

[118] Apache, "Tomcat Server," http://tomcat.apache.org.

[119] S. Taoka, D. Takafuji, T. Iguchi, and T. Watanabe, "Performance Comparison of Algorithms for the Dynamic Shortest Path Problem," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E90-A, no. 4, pp. 847–856, 2007.

[120] C. Demetrescu and G. F. Italiano, "Experimental analysis of dynamic all pairs shortest path algorithms," *ACM Trans. Algorithms*, vol. 2, no. 4, pp. 578–601, 2006.

[121] P. Narvaez, S. Kai-Yeung, and T. Hong-Yi, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 6, pp. 734–746, Dec 2000.

[122] B. Xiao, J. Cao, Q. Zhuge, Z. Shao, and E. Sha, "Dynamic update of shortest path tree in OSPF," in *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, May 2004, pp. 18–23.

[123] B. Xiao, J. Cao, and Q. Lu, "Dynamic SPT update for multiple link state decrements in network routing," *Journal of Supercomputing*, vol. 46, no. 3, pp. 237–256, Jan 2008.

[124] V. Eramo, M. Listanti, and A. Cianfrani, "Implementation and Performance Evaluation of a Multi-Path Incremental Shortest Path Algorithm in Quagga Routing Software," in *Proceedings of the 7th International Workshop on Design of Reliable Communication Networks*, October 2007.

[125] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," May 1990, RFC 1157.

[126] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, "Introduction to version 2 of the Internet-standard Network Management Framework," April 1993, RFC 1441.

[127] R. Presuhn, "Version 3 of the Protocol Operations for the Simple Network Management Protocol (SNMP)," December 2002, Standards Track RFC 3416.

[128] U. Blumenthal and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)," December 2002, Standards Track RFC 3414.

[129] B. Wijnen, R. Presuhn, and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)," December 2002, Standards Track RFC 3415.

[130] K. McCloghrie, D. Perkins, and J. Schoenwaelder, "Structure of Management Information version 2 (SMIv2)," April 1999, Standards Track RFC 2578.

[131] D. Levi, P. Meyer, and B. Stewart, "RFC3413: Simple Network Management Protocol (SNMP) Applications," December 2002, Standards Track RFC 3413.

[132] C. Chiang, S. Demers, P. Gopalakrishnan, L. Kant, A. Poylisher, Y. H. Cheng, R. Chadha, G. Levin, S. Li, and Y. Ling, "Performance analysis of DRAMA: a distributed policy-based system for MANET management," in *Proceedings of the Military Communications Conference (MILCOM)*, 2006.

[133] H. Choi, N. Kim, and H. Cha, "6lowpan-snmp: Simple network management protocol for 6lowpan," in *Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications*, 2009.

[134] A. Jacquot, J. Chanet, K. Hou, X. Diao, and J. Li, "A new approach for Wireless Sensor Network management: LiveNCM," *New Technologies, Mobility and Security (NTMS)*, 2008.

[135] G. Kuthethoor, P. Sesha, J. Strohm, P. OÕNeal, G. Hadynski, D. Climek, J. DelMedico, D. Kiwior, D. Dunbrack, D. Parker *et al.*, "Performance analysis of SNMP in airborne tactical networks," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, 2008.

[136] R. Badonnel, R. Festor, O. Team, L. Lorraine, and V. les Nancy, "Probabilistic Management of Ad-Hoc Networks," in *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006, pp. 339–350.

[137] V. Pacheco and R. Puttini, "An administration structure for the OLSR protocol," in *Proceedings of the International Conference on Computational Science and Its Applications*, 2007.

[138] R. Badonnel *et al.*, "Management of mobile ad hoc networks: information model and probe-based architecture," *International Journal of Network Management*, vol. 15, no. 5, pp. 335–347, 2005.

[139] R. G. Cole, J. Macker, and A. Morton, "Definition of Managed Objects for Performance Reporting," February 2011, Internet Draft, work in progress, draft-ietf-manet-report-mib-01.txt.

[140] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc. 35*, pp. 99–109, 1943.

[141] L. Andrey, O. Festor, A. Lahmadi, A. Pras, and J. Schönwälder, "Survey of SNMP performance analysis studies," *International Journal of Network Management*, vol. 19, no. 6, pp. 527–548, 2009.

[142] R. Enns, "NETCONF Configuration Protocol," Dec. 2006, Standards Track RFC 4741.

[143] "SNMP4J webpage," http://www.snmp4j.org.

[144] U. Blumenthal, F. Maino, and K. McCloghrie, "The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model," June 2004, Standards Track RFC 3826.

[145] X. Wang and H. Yu, "How to break MD5 and other hash functions," *Advances in Cryptology–EUROCRYPT 2005*, pp. 19–35, 2005.

[146] "Wireshark webpage," http://www.wireshark.org.

[147] A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of applied cryptography.* CRC, 1997.

[148] T. Clausen and K. Mase, "Link Buffering for MANETs," October 2004, Internet Draft, work in progress, draft-clausen-manet-linkbuffer-00.txt.

[149] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE std*, June 2007.

[150] C. Hutzler, D. Crocker, P. Resnick, E. Allman, and T. Finch, "RFC5068: Email Submission Operations: Access and Accountability Requirements," November 2007, Best Current Practice RFC 5068.

[151] J. Joseph, A. Das, and B. Lee, "Security Threats in Ad Hoc Routing Protocols," *Guide to Wireless Ad Hoc Networks*, pp. 1–18, 2009.

[152] N. Komninos, D. Vergados, and C. Douligeris, "Security for Ad Hoc Networks," *Handbook of Information and Communication Security*, pp. 421–432, 2010.

[153] C. Sreedhar, S. Verma, and N. Kasiviswanath, “A Survey on Security Issues in Wireless Ad hoc Network Routing Protocols,” *International Journal on Computer Science and Engineering (IJCSE)*, vol. 2, no. 2, pp. 224–232, 2010.

[154] P. Yi, Y. Wu, F. Zou, and N. Liu, “A Survey on Security in Wireless Mesh Networks,” *IETE Tech Rev*, pp. 6–14, 2010.

[155] C. Adjih, D. Raffo, and P. Mühlethaler, “Attacks against OLSR: Distributed key management for security,” in *Proceedings of the OLSR Interop and Workshop*, 2004.

[156] A. Fourati, H. Badis, and K. Al Agha, “Security Vulnerabilities Analysis of the OLSR Routing Protocol,” in *Proceedings of the 12th International Conference on Telecommunications, ICT*, 2005.

[157] B. Kannhavong, H. Nakayama, N. Kato, A. Jamalipour, and Y. Nemoto, “A study of a routing attack in OLSR-based mobile ad hoc networks,” *International Journal of Communication Systems*, vol. 20, no. 11, pp. 1245–1261, 2007.

[158] Y. Li, C. Wang, W. Zhao, and X. You, “The Jamming problem in IEEE 802.11-based mobile ad hoc networks with hidden terminals: Performance analysis and enhancement,” *International Journal of Communication Systems*, vol. 22, no. 8, pp. 937–958, 2009.

[159] C. Adjih, E. Baccelli, T. Clausen, P. Jacquet, and G. Rodolakis, “Fish Eye OLSR Scaling Properties,” *Journal of communication and networks*, vol. 6, no. 4, pp. 343–351, 2004.

[160] L. Viennot, “Complexity results on election of multipoint relays in wireless networks,” INRIA, Tech. Rep., 2007, Research report RR-3584.

[161] B. Kaliski and J. Staddonh, “PKCS 1: RSA Cryptography Specifications Version 2.0,” October 1998, Informational RFC 2437.

[162] National Institute of Standards & Technology, “Digital Signature Standard,” June 2009, FIPS PUB 186-3.

[163] D. Johnson, A. Menezes, and S. Vanstone, “The elliptic curve digital signature algorithm (ECDSA),” *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[164] H. Krawczyk and M. Bellare, “HMAC: Keyed-Hashing for Message Authentication,” February 1997, Informational RFC 2104.

[165] G. Pottie and W. Kaiser, “Wireless integrated network sensors,” *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, 2000.

[166] M. Goyal, E. Baccelli, A. Brandt, R. Cragie, and J. Martocci, “Reactive Discovery of Point-to-Point Routes in Low Power and Lossy Networks,” May 2011, Internet Draft, work in progress, draft-ietf-roll-p2p-rpl-03.

[167] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*, 2004.

[168] J. Hui, J. Vasseur, D. Culler, and V. Manral, "An IPv6 Routing Header for Source Routes with RPL," March 2011, Internet Draft, work in progress, draft-ietf-6man-rpl-routing-header-03.

[169] SICS, "ContikiRPL," http://www.sics.se/contiki.

[170] Berkeley University, "OpenWSN," http://openwsn.berkeley.edu/wiki/OpenRpl.

[171] T. Clausen, G. Hansen, L. Christensen, and G. Behrmann, "The Optimized Link State Routing Protocol, Evaluation through Experiments and Simulation," in *Proceedings of the IEEE Symposium on Wireless Personal Mobile Communications*, October 2001.