Technische Universität München
Fakultät für Informatik
Netzwerkarchitekturen
Prof. Anja Feldmann (PhD)

Diplomarbeit in Informatik

Autoconfiguration of Mobile Ad Hoc Networks

Ulrich Herberg

Technische Universität München
Fakultät für Informatik
Netzwerkarchitekturen
Prof. Anja Feldmann (PhD)

Diplomarbeit in Informatik

Autoconfiguration of Mobile Ad Hoc Networks

Eingereicht von: *Ulrich Herberg*
ulrich@herberg.name

Abgabedatum: *15. Mai 2007*

Aufgabenstellerin: *Prof. Anja Feldmann (PhD), TUM*
Betreuer: *Prof. Thomas Clausen (PhD), Ecole Polytechnique*

Erklärung der Selbstständigkeit

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Mai 2007

Ulrich Herberg

Acknowledgement

This thesis has been created in close collaboration with Thomas Clausen to whom I wish to express my gratitude. Special thanks also to Emmanuel Baccelli, Philippe Jaquet, Cedric Adjih for their discussions and comments. Thanks also to Professor Anja Feldmann who accepted me to write my thesis at Ecole Polytechnique.

Zusammenfassung

In dieser Diplomarbeit wird eine Lösung zur konfigurationsfreien Allokation von IP Addressen in mobilen Ad-hoc Netzen (MANET) vorgeschlagen. Aktuelle Vorschläge aus der Literatur basieren auf einem Architekturmodell von MANETs, das MANET Knoten als Hosts im selben Subnet betrachtet. Unserem Verständnis nach führt dies jedoch zu einer Inkompatibilität mit der IP Infrastruktur des Internets. Folglich können MANET Knoten nicht korrekt in das Internet integriert werden. In der vorliegenden Arbeit wird ein Protokol spezifiziert, das kohärent mit einem Architekturmodel von MANETs ist, welches MANET Knoten als Router mit verbundenen Hosts betrachtet. Außerdem hat dieses Protokoll sehr wenige Voraussetzungen im Vergleich zu aktuellen Lösungen, da es werder auf Link-lokalen IP Adressen noch auf einem Multi-Hop Routing Protokoll basiert. Das Protokoll wurde formal auf Korrektheit validiert und in einer realen Testumgebung sowie für den Netzwerksimulator NS2 implementiert. Schließlich wurden Optimierungen und Erweiterungen des Protokolls vorgeschlagen und eine Performance-Analyse durchgeführt.

Abstract

In this thesis, a solution for autoconfiguring IP addresses of mobile ad-hoc networks is proposed. Current proposals are based on an architectural model of MANETs considering MANET nodes as hosts all being in the same subnet. However, it is our understanding that this leads to an incompatibility with the current IP infrastructure. Consequently, MANET nodes cannot be correctly integrated into the Internet. A protocol is specified in this thesis which is coherent to an architectural model considering MANET nodes as routers with possibly attached hosts. Thus, the protocol can be correctly integrated in the current IP infrastructure of the Internet. Additionally, this protocol has very little prerequisites in comparison to current solutions as it does not depend on link-local addresses or a multi-hop routing protocol. The protocol is formally validated for correctness and implemented in a real-life testbed as well as for the NS2 network simulator. Finally, optimizations and extensions of the protocol are proposed and a short performance analysis is presented.

Contents

List of Figures

List of Tables

1 Introduction

In the last few years, the use of wireless networks has grown on a large scale. The 802.11 wireless network standard[3] started with a modest bandwidth of 1 and 2 Mbps in 1999. In 2003, a maximum bandwidth of 54 Mbps has been standardized. While the speed of wireless networks has largely increased, the prices for a WiFi card have lowered. For these two reasons, and due to an enormous increase in public wireless hotspots, more and more devices include a wireless interface by default. All modern laptops and some recent mobile phones are equipped with a wireless device. IDC estimates the global market size of 802.11 components[1] as $1.5 billion in 2005 and an estimated $3.2 billion in 2010, a 17% compound annual growth rate (CAGR)[2].

Due to the reasons stated above, it is not astonishing that a lot of research in this area is performed. Especially in the area of mobile ad-hoc networks (MANET), researchers are quite active. A MANET is a network of wireless devices that communicate over multi-hop wireless links. These networks may dynamically and spontaneously change their topology which makes research more difficult. While the research community has been very active in the field of routing in MANETs, much less progress has been made in the area of autoconfiguration which is the process of a node getting an IP address or prefix upon initialization. In the scope of this thesis, existing autoconfiguration approaches will be analyzed and categorized, and then an autoconfiguration algorithm for MANETs will be specified and analyzed. In particular, this algorithm will be based on a new architectural model of MANETs which is described in this thesis as well.

This thesis was written in the Hipercom group / LIX (Laboratoire d'Informatique de l'X) at Ecole Polytechnique, Palaiseau, France under the guidance of Professor Thomas Clausen (Ecole Polytechnique) and the formal supervision of Professor Anja Feldmann (TUM).

1.1 Problem description

Unlike wireless networks with central access points and DHCP servers, stateless mobile ad hoc networks are completely decentralized and possess no central infrastructure. In addition, due to the limited range of a mobile node, a node normally has no complete view of the MANET. Thus it is necessary to have autoconfiguration algorithms, i.e. algorithms for assigning an IP address to a node upon connection to a MANET that works under these constraints. These addresses have to be unique in a certain scope to avoid packets being sent to wrong nodes. A task of particular difficulty is the address uniqueness test: as the nodes normally do not have a complete view of all participating nodes in the network, they cannot determine a duplicate address conflict offhand. So far, there are several possible solutions for this problem, but none of these has been standardized. One reason for this might be the lack of a clear classification and terminology. There are papers surveying the different approaches [84], but none of them uses a clear classification. Also, terminology is often used in different or even contradictory ways. Another problem is that all papers so far have a common understanding about a MANET which might not reflect the equivalent of a wired LAN. This architectural

1 This includes 802.11b, 802.11a, 802.11g, dual-band 802.11a+b/g chipsets, 802.11n, and "pre-n" and "draft-n" components shipping into client-side applications and access point devices

2 http://www.idc.com/getdoc.jsp?containerId=203617

misunderstanding leads to an incompatibility of current solutions with the commonly agreed IP infrastructure of the Internet.

1.2 Purpose of the thesis

The purpose of this thesis is to address the above-mentioned issues by developing a coherent classification of "the autoconfiguration problem" by defining goals, subgoals and approaches to fulfill these subgoals. Then a terminology is given, trying to standardize words and to classify them as well in an abstract way. Next, a comparison between existing solutions is delivered including their classification in the scheme.

The second part of the thesis proposes a specification of an autoconfiguration algorithm which is intended to correspond to an architectural model which is coherent with the Internet. In addition, the solution is based only on very few assumptions. While other proposed algorithms in literature are mostly suitable only for certain cases, the algorithm presented in this thesis should be runnable in all kind of MANETs. After analyzing the algorithm, functionality and performance tests are performed. An automated real-life test with several Nokia 770 wireless devices shows the advantages and possible issues with the algorithm. A simulation provides scalability tests and a formal verification validates requirements for an autoconfiguration algorithm.

Out of the scope of this thesis are partitioning and merging of MANETs, and any security considerations of MANET routing protocols or the autoconfiguration algorithm. In addition, no complete overview of all MANET routing protocols is presented but only a compact overview of the most important proactive and reactive routing protocols.

1.3 Structure of the master's thesis

In **chapter 1** a short introduction to wireless networking is given, followed by a brief description of the problem of autoconfiguration. The purpose of this thesis – a classification of current autoconfiguration algorithms and introducing a new autoconfiguration algorithm – is presented before this overview of the structure.

Chapter 2 the status of current research in the field of MANETs is presented. Starting with a general presentation of MANETs, two different types of routing are introduced, followed by a more detailed description of OLSR, a proactive protocol developed at Hipercom. At the end of the chapter, a short introduction to autoconfiguration issues in MANETs is presented.

Chapter 3 presents a classification and terminology of autoconfiguration issues in MANETs. This classification will be used in the related work chapter.

Chapter 4 is a survey of related work on autoconfiguration algorithms. The most known algorithms are classified and described, also mentioning their special use cases and their deficiencies in terms of coherence with the Internet architecture.

In **Chapter 5** an autoconfiguration algorithm is presented which aims to be coherent with the architectural model of the Internet. After outlining the general purpose and design goal of the algorithm, the protocol is specified beginning with the configuration of link-local addresses. Finally, router and host configuration protocols are presented.

In chapter 6 extensions and special issues of the autoconfiguration protocol of chapter 5 are proposed and analyzed.

Chapter 7 validates formally the autoconfiguration protocol using the model checker UPPAAL.

Chapter 8 explains the testbed written in C++ for the Nokia 770 mobile devices. Testing results are presented in this section.

Chapter 9 presents the simulation of the autoconfiguration algorithm in the NS2 simulator and shows results of the tests in terms of correctness of the protocol and a performance comparison between the basic protocol and one optimization.

Chapter 10 is the final conclusion about the algorithm, summarizing the methodology applied in this thesis, and the results of the work, finalized by a prospective of further work.

2 Motivation

Wireless local area networks (WLAN) and other mobile networks, like GSM for mobile phones, use radio waves to communicate to each other in a certain limited range. Typically, wireless devices are connected to a central access point which is handling the routing of the packets among the nodes. Wireless nodes operating in ad-hoc mode, however, communicate directly, without the necessity of a central access point. 802.11 ad-hoc connections are typically single hop connections, allowing only two devices to exchange packets due to the lack of routing among the nodes.

Multi-hop networks allow nodes to route packets over several hops from the source node to the destination node. Mobile ad hoc networks (MANET) are spontaneously changing, chaotic and autonomous multi-hop networks, allowing organization and routing to be done automatically by the network itself. The main problem discussed in this thesis is the initialization phase of node, i.e. the phase when a new node enters the MANET. It has to get an IP address in order to communicate with the other nodes and it has to be assured that no two nodes in a certain scope have the same IP address.

2.1 IETF MANET working group

Created in 1997, the MANET working group aims to develop 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." [90]

So far, several Internet drafts about different routing protocols have been created, but only four of them have become Request For Comments (RFC)[1]: *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations* (RFC2501) [29], *Ad Hoc On Demand Distance Vector (AODV) Routing* (RFC 3561) [68], *Optimized Link State Routing Protocol* (RFC 3626) [24] and *Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)* (RFC 3684) [65] Other protocols like NHDP, DYMO, OLSRv2 and SMF are likely to be sent to the Internet Engineering Steering Group for standardization in 2007. The IETF standardization process is documented in details in RFC 2026 [11].

2.2 Definition of a MANET

A mobile ad hoc network (MANET) is an autonomous system of mobile nodes that are free to move arbitrarily. The nodes are equipped with antennas for receiving and transmitting data frames. At a given point in time, depending on the nodes' positions and their transmitter and receiver coverage patterns, transmission power levels and co-channel interference levels, a wireless connectivity in the form of a random, multi-hop graph or "ad hoc" network exists between the nodes [29]. MANETs can either be stand-alone networks or may be connected by a gateway to other wired networks (like the Internet) or other wireless networks.

1 RFCs are documents that have passed an IESG review and are published by the IETF. They can be found at http://www.ietf.org/rfc.html

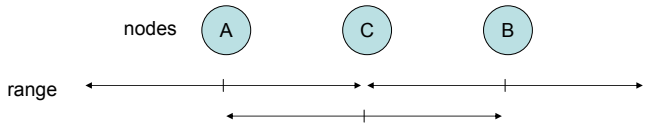


Figure 2.1: Node C can route packets from A to B

Table 2.1: Overview of the 802.11 family (Source: http://en.wikipedia.org/wiki/802.11)

Protocol	Release Date	Op. Frequency	Data Rate (Max)	Range (Indoor)
Legacy	1997	2.4 GHz	2 Mbit/s	?
802.11a	1999	5 GHz	54 Mbit/s	30 meters
802.11b	1999	2.4 GHz	11 Mbit/s	50 meters
802.11g	2003	2.4 GHz	54 Mbit/s	30 meters
802.11n	2007 (projected)	2.4 GHz or 5 GHz	540 Mbit/s	50 meters
802.11s	2008 (projected)			

MANETs have two main attributes:

- **Dynamic topology**: In contrast to wired networks, the topology of ad hoc networks may change at any time. Nodes move around, get out of the range of other wireless nodes, leave or join a MANET randomly and unpredictably. Links between nodes may be bidirectional or unidirectional. Also whole parts of MANETS may migrate into another MANET, and MANETs can be partitioned.
- **Bandwidth-constrained**: Typically, wireless connections deliver much less bandwidth than their wired counterparts. In addition, background noise or other environmental influences can limit the bandwidth. This causes problems when one wants to assure a certain level of quality, for example for voice applications. The particularity of wireless networks is that all nodes use the same communication channel. So adjacent nodes can reduce their bandwidth even by not talking directly to each other but with one of their neighbors each.

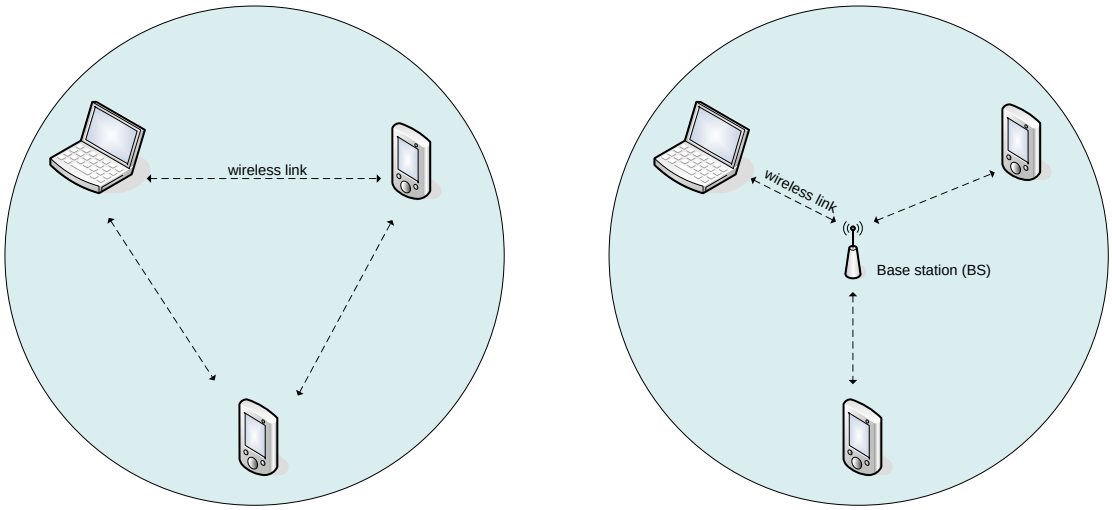
Consider the following example of three wireless nodes, Alice, Bob and Carol, as depicted in figure 2.1. Alice and Bob would like to communicate with each other but they are out of reach. They cannot "hear" each other. Carol however hears them both as long as they do not "speak" at the same time. In order to Bob say hello to Alice, Carol must forward his message – this would be a typical case in a multi-hop network.

2.2.1 802.11 wireless interfaces

Lots of the particular properties of a mobile ad hoc network are due to the special attributes of wireless interfaces. These attributes are standardized by the group 11 of the IEEE LAN/MAN Standards Committee (IEEE 802). Currently, the 802.11 family includes six over-the-air modulation techniques. 802.11b and 802.11g standards use the 2.4 gigahertz (GHz) band, while 802.11a uses the 5.0 GHz band, which is less affected by interferences. See table 2.1 for an overview of the different standards in the 802.11 family.

The 802.11 standard defines two modes for connecting to other nodes [3]:

- **Infrastructure mode**: The basic service set (BSS) – also called infrastructure mode



(a) IBBS mode (b) BBS mode

Figure 2.2: 802.11 modes

– is the basic building block of an IEEE 802.11 LAN. Nodes within a BSS communicate via a base station (BS).

- **Ad hoc mode**: This is an informal labeling for independent basic service sets (IBSS), which is the most basic type of 802.11 LAN. It allows stations in mutual communication range to exchange packets without the necessity of a base station. Because this type of IEEE 802.11 LAN is often formed without pre-planning, for only as long as the LAN is needed, this type of operation is often referred to as an *ad hoc network*.

Throughout this thesis, it is assumed that wireless nodes operate in ad-hoc mode. A more detailed description of the 802.11 family can be found in [63].

2.2.2 Neighborhood

A particularity of wireless networks is the different unique partial view of the MANET of each wireless interface. Consider the example in figure 2.1: Node A would only have node B as his neighbor, node B sees both nodes A and B, and node C reaches only node B. So in contrast to wired networks where each node has the same view of the network on the same link, wireless interfaces do not have a complete view of the network. Each of them may have a particular neighborhood, which afflicts routing and autoconfiguration algorithms.

Another special issue of wireless networks is that nodes send and receive packets on the same interface. On MANET routers with several adjacent nodes, duplicate IP messages may occur. Routers receiving a unicast message must ignore the packet if they are not the designated next-hop node as stated in the header of the packet. For multicast or broadcast packets, special mechanisms have to be deployed to avoid infinite loops of packets (like keeping track of sequence numbers of already received packets).

Furthermore, links between adjacent nodes may be unidirectional. Due to the nature of wireless connections, a node A may receive packets from node B but not vice versa. These links are called “unidirectional” in contrast to bidirectional links where both adjacent nodes can communicate with each other. MANET routing protocols can detect unidirectional links which are called asymmetric in contrast to symmetric links. This terminological difference is due to the fact that at a given time, a routing protocol may

not yet have detected a bidirectional link as symmetric (see also the terminology in section 3.2). Most routing algorithms ignore asymmetric links as they are not trivial to handle. A discussion of the problems of unidirectional links can be found in [7].

2.3 Applications of MANETs

Wireless ad hoc networks were only a small niche topic of research some years ago. With researchers having developed standardized and well tested routing algorithms, MANETs have become a domain of great interest for industrial applications. Companies in the mobile phone industry and the automotive industry declared interest in MANETs and launched cooperations with research groups like Hipercom.

There are several possible applications of MANETs which can be interesting and replace existing solutions with 3G components due to the lower costs. One example would be inter-car communication. Cars in dense traffic areas could send multicast messages in a certain area containing information about traffic jams. Also Internet connections could be shared, as long as at least one car in the MANET acts as a router (either by having access to a base station or by using a 3G mobile phone).

Another possible application would be using VoIP on mobile phones without the necessity of a nearby base station. Adjacent mobile phone users would simply route the VoIP packets to their destination or at least to the next base station. For customers this could on the one hand decrease costs for GSM / UMTS connections. On the other hand, it would simplify getting a connection as MANETs are self-configuring. While for base stations, a manual connection has to be established with handshaking protocols and assigning IP addresses, the user would not have to care about that in MANETs.

There are lots of other possible applications with MANETs, some of them being mentioned in [7].

2.4 MANET routing protocols

A crucial factor for a MANET is the underlying routing algorithm. The algorithm has to handle the fast-changing topology on the one hand, while keeping overhead as small as possible on the other hand. There are mainly two classes of routing algorithms, proactive and reactive protocols [25]. There are other less known classes of routing algorithms, which are not considered in this thesis[2].

Nodes in **reactive protocols** acquire the routing information on demand. Whenever a node wants to send a packet to another node in the MANET, it floods the network to find a suitable route for the packet. So a node does normally not have a complete view of the network, but just gets a route to a destination on demand. This reduces the number of packages (and thus the overhead) having to flood the whole network in order to get to know all other nodes. On the other hand, each time when a node needs to send a packet to another node, it costs a certain time to acquire the route. An example of a reactive protocol is "Ad-hoc On-Demand Distance Vector" (AODV) [68]. Reactive protocols are designed with the following premises (taken from [7]). Note that these are design goals of reactive protocols claimed by authors of reactive protocols and are not discussed or commented in this thesis.

- Network locality is strong: most active routes are topologically local, within one or two hops

2 See `http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list` for a list

- The application can work around occasional routing glitches if recovery is expedited
- While routing may change continuously in the global sense, individual routes generally survive long enough to perform common application tasks
- The overhead of searching for a route when it is needed (which may take several round trip times) is acceptable.
- The ration of multi-hop routes actively being discovered and maintained is small compared to the number of such possible routes within a MANET area.

In **proactive protocols** however, nodes keep track of the complete network by acquiring a complete routing table on a regular basis. When a node wants then to send a packet, it does not lose time for getting a new route first. But then the overhead flooding the net constantly is considerably higher. "Optimized Link State Routing Protocol"(OLSR) [24] is the most known proactive protocol so far. Proactive protocols have the following design goals [7]:

- Network locality is indeterminate; routes of any length may be commonly used, or not at all.
- The application can work around occasional routing glitches , but recovery must be almost immediate.
- The overhead of calculation and information flooding is acceptable, but the overhead of searching is not.
- Delays of packet delivery are unacceptable; proactive protocols use the best-effort strategy to deliver packets with the smallest loss of time possible.

There is also some research done in the area of hybrid protocols which seek to combine the approaches of both reactive and proactive protocols. However, this research is in the early stages.

In the following subsections, the most known routing protocols – AODV and OLSR – are presented in more detail.

2.4.1 Optimized Link State Routing Protocol (OLSR)

Defined in RFC 3626 [24], OLSR is similar to the standard routing protocol OSPF [61] but more optimized for wireless ad hoc networks. As introduced in section 2.4, OLSR is a proactive routing protocol for MANETs developed at Hipercom.

OLSR is mainly based on three concepts: Neighbor sensing, an efficient distribution mechanism for generic messages and an efficient way to distribute topology information among the nodes. In the following, these mechanisms will be explained.

Neighbor sensing

Nodes have to detect their direct neighbors, to which they have a bidirectional link. Due to the nature of wireless connections, links may become unidirectional or even invalid. Thus nodes have to check these neighborhood links regularly.

In addition to the *one-hop neighbors*, the concept of *two-hop nodes* has been introduced in OLSR. A two-hop neighbor of a node A is a node which has a symmetric link to a symmetric neighbor of A and which is not node A itself [25].

The aim of neighbor sensing in OLSR is to be completely independent from the underlying link layer. For example, OLSR can be used based on 802.11 link layer, but also on Bluetooth or wired networks. While some information of the link layer – like the link quality or link-layer event notifications – *may* be used for OLSR, it is not mandatory.

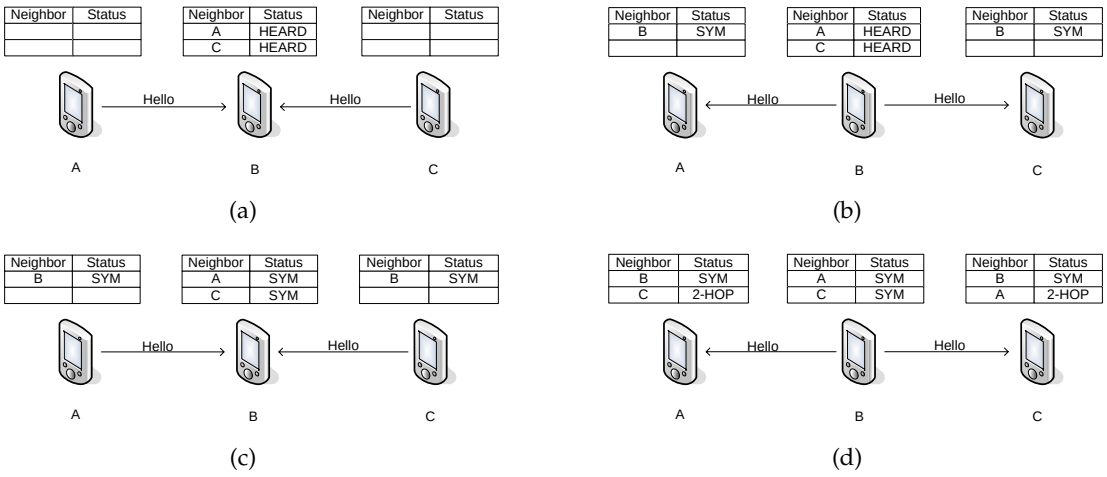


Figure 2.3: OLSR neighbor detection

In order to get all neighbors and their link status, nodes send so-called HELLO packages to all their direct neighbors using the broadcast mode. These packets contain the source address of the node and a list of the neighbors of a node and their link status. When a node receives a HELLO packet from another node, it can derive all one-hop and two-hop neighbors and the link status. If a node A receives a HELLO packet from node B including node A in the neighbor list of B, the link between A and B is bidirectional, otherwise it is unidirectional.

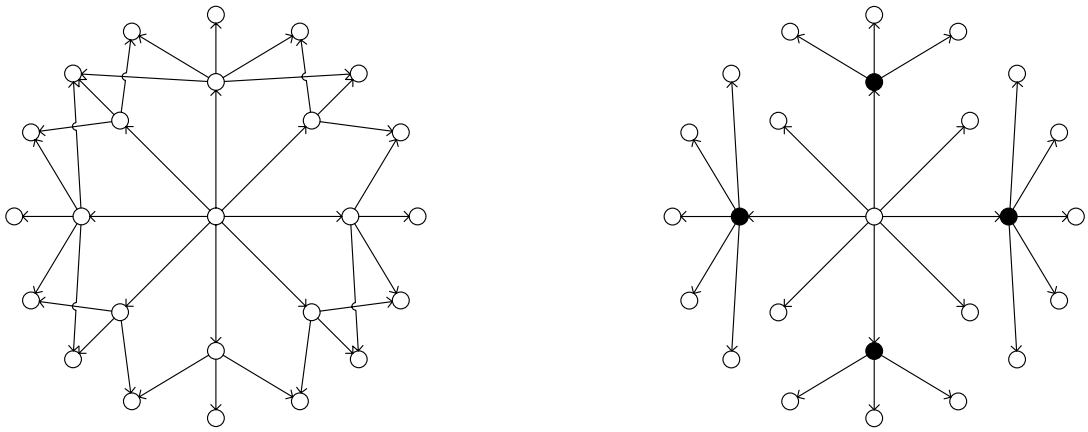
Each node stores information about its one- and two-hop neighbors in a table, which has to be updated on a regular basis. If an entry is not updated after its expiration time, it will be deleted from the table.

In figure 2.3 an example of OLSR neighborhood discovery is depicted. In the first step (figure 2.3(a)) nodes A and C send HELLO messages to their common neighbor B. Thus B knows about A and C but the link is not yet acknowledged as symmetric (it might be that A and C can send packets to B but not vice versa). In the next step (figure 2.3(b)) B sends back HELLO messages to A and C so that they both know that their link to B is symmetric. When they send back another HELLO message to B (figure 2.3(c)), B also knows about the symmetric link. In the final figure 2.3(d)) includes the addresses of all its symmetric neighbors in the HELLO messages. Thus, nodes A and C get to know each other as 2-hop neighbors.

Generic message diffusion

Now that each node has a mechanism to reach all its neighbors up to two hops, an information distribution mechanism for bigger MANETs is needed. Topological information about all nodes in the MANET have to be broadcasted through the MANET in an efficient way. Flooding the topological information to all nodes would inflict lots of packets to be received several times by a node (as depicted in figure 2.4(a)). Thus, OLSR includes an algorithm, which efficiently broadcasts information through the MANET using so-called Multipoint Relay (MPR) sets.

As stated in section 2.4, the aim of any routing protocol is (i) to reach all nodes and (ii) to require as little overhead as possible. A classic flooding algorithm fulfills requirement (i), as it reaches all nodes. Nodes only have to delete duplicate packets by storing a table of received packets with their source address and sequence number. Duplicate packets can then simply be discarded by comparing source address and sequence number of an arriving packet with the entries of this table. OLSR however also fulfills condition (ii)



(a) Classic flooding

(b) Optimized broadcasting using MPR relaying

Figure 2.4: Broadcasting without and with MPR relaying

by implementing a more efficient algorithm than classic flooding. Flooding in wireless environments has a large impact on performance of the network, since wireless interfaces receive and send packets on the same channel. Whenever a node receives a packet over a wireless interface, it has to wait until end of receipt before sending a packet itself.

Instead of relaying packets to all neighbors, OLSR uses only a subset of next hop nodes, called the MPR set. Each node has a set of MPR selectors, i.e. nodes that have selected it as MPR. For nodes which do not relay any packets, the MPR selector set is empty. There are several possible techniques to select the minimal subset of MPRs, some of them proposed in [70]. An optimal MPR set is displayed in the example depicted in 2.4(b).

Topology information

In order to allow nodes to send packets to any other node in the MANET, the topology has to be broadcasted to all nodes. All nodes having a non-empty MPR selector set broadcast so-called topology control (TC) messages containing the nodes' MPR selector set to the MANET. These TC messages are similar to the link-state advertisement messages in OSPF [61]. In a TC message, the source address of the node is included as well as all MPR selectors of the node. By the distribution of TC messages, nodes gain a partial topological view of the MANET. From this partial view, they can defer a complete routing table using any shortest path algorithm.

2.4.2 Ad-hoc On-Demand Distance Vector (AODV)

AODV is defined in RFC 3561 [68]. As a typical reactive protocol, nodes running AODV acquire a route to a destination node only on demand. In AODV there are three principal message types: route requests (RREQ), route replies (RREP) and route errors (RERR).

Whenever a node wants to send a packet to a so far unknown node, it broadcasts a RREQ packet to all its neighbors. Upon receipt of a RREQ packet, a node examines its route table and looks for a "fresh" (i.e. not yet expired) entry of the destination host. If it does not find one, it retransmits a RREQ to all its neighbors, storing a unicast route back to the originating node in its routing table. If the node however finds a non-

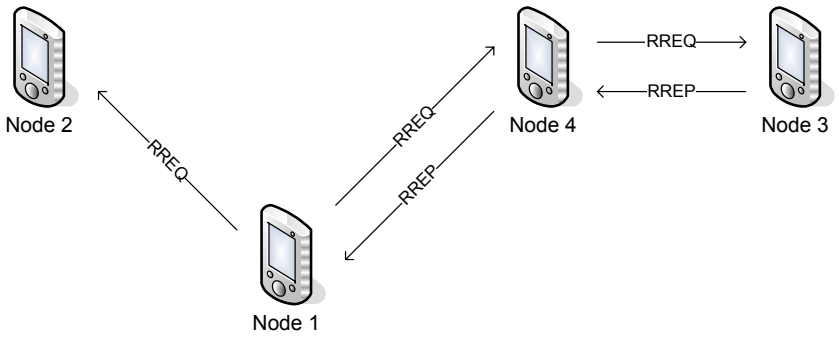


Figure 2.5: AODV example

expired entry in its routing table or is the destination node itself, it responds with a RREP packet to the unicast address of the originating node of the RREQ packet.

A simple example of the functionality of AODV is depicted in figure 2.5. Node 1 wants to send a packet to node 3, but has no entry in its routing table. In order to reach node 3, node 1 will broadcast a RREQ packet to all its neighbors (nodes 2 and 4). The RREQ message contains the source address, the destination address, the lifespan of the message and a sequence number. The receiving nodes than have the following choice: if they do not know a route to node 4 (like for example node 2), the RREQ packet will continuously being re-broadcasted until it reaches its end of life. Rebroadcasting the same packet twice is impossible as all nodes keep a list of source address and sequence number of all packets. In the example, the RREQ will reach node 3, so it will reply with a RREP packet to the unicast address of node 1. If node 4 had already a route to node 3 (e.g. if it had sent a packet to it before), node 4 can reply in favor of node 3 to reduce network congestion.

When a node discovers that a link to one of its neighbors is broken, it will send an error packet (RERR) to the nodes having used this link. These nodes will then delete the entry from their routing table and send a new RREQ packet in order to discover another route to the destination.

AODV has some extensions, among of them an "expanding ring" flooding [25]. Route requests have a limited time-to-live (TTL) upon sending. On each hop, the TTL is reduced by one. When a node receives a packet and reduces its TTL to zero, it will discard the packet. If the original sender of the RREQ thus does not receive any answer (i.e. a RREP), it will resend another RREQ with a higher TTL, so that the packet gets broadcasted further in the MANET.

2.5 Comparison of MANETs to wired LANs

2.5.1 Architecture misperception

The structure of MANETs is in principle similar to wired LANs and is based on the same network protocols in the ISO/OSI model. However, in most papers about MANETs a clear definition of the architecture including a comparison to the wired case is missing. This section shall recall the basics of Local Area Networks and compare it to the architecture of MANETs. The considerations are based on [13] and [16].

Figure 2.6 depicts a classical IP link. Several hosts share the same link which means that the can communicate with each other on the link-layer basis. For example, they could be connected via a switch or hub. Possibly, one router R may be attached to the link

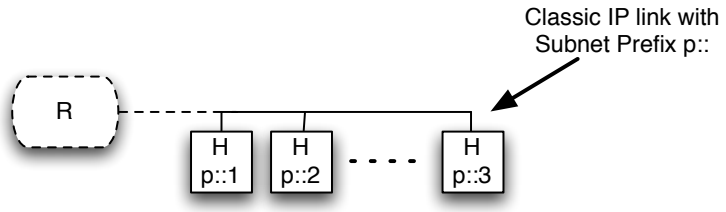


Figure 2.6: Classic IP link model: hosts(H) connected to the same link have assigned IP addresses from a common prefix p

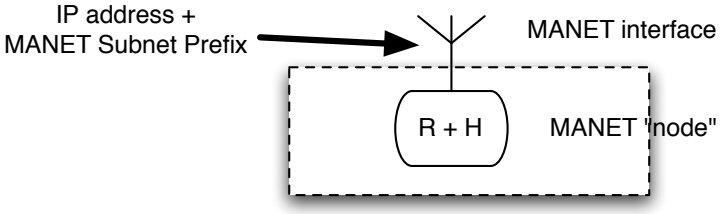


Figure 2.7: Misperception of a MANET node as combined router and host

which assigns a prefix to the nodes. All nodes on the link having the same prefix may directly communicate to each other without routing (refer to [40] for more information about IP addressing). They form a subnet.

Most of the MANET papers consider a MANET node as depicted in figure 2.7. They regard a node as a combined router and host with on or several MANET interfaces. Consequently, they interpret a MANET as a subnet with all nodes sharing the same prefix (see figure 2.8).

The problem with this view of a MANET is the definition of the “link”. In a classical wired LAN all hosts on a link can directly communicate with all nodes in the same subnet. This is **not** possible in the described view of a MANET. Without routing, only nodes in the direct neighborhood can communicate. This contradicts the commonly shared prefix in the MANET subnet.

In this thesis, MANET nodes are considered as depicted in figure 2.9. A node is basically a subnet itself with one router and zero or more hosts. These hosts might be either internal hosts (i.e. applications running on the node or virtual hosts using VMware or similar solutions) or they may be externally attached hosts (e.g. by a switch). The corresponding MANET would look as shown in figure 2.10.

The following problems may arise when using the "wrong" architectural model (extracted from [16]:

- `Routing incompatibility:`
  Forwarding of IP datagrams within the MANET will prompt intermediate nodes to produce ICMP redirects. This is appropriate since IP datagrams delivered within a subnet are not supposed to be forwarded by a router since a direct link between any two nodes within a subnet is supposed to exist.

- `Incompatibility with other protocols and applications:`
  The routing incompatibility problem is often masked by disabling ICMP redirect messages. But this can be considered as disabling a symptom of an incorrect

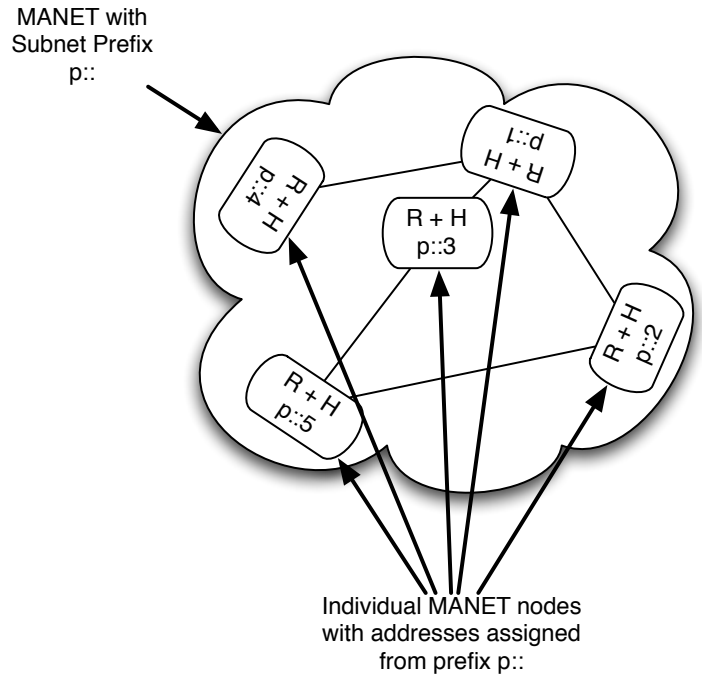


Figure 2.8: Misperception of a MANET: a MANET as one subnet sharing the same prefix p

network model for a single application (routing) only, and leads to the specific and reasonable question if other applications and protocols require similar tweaks.

Concluding, the properties of the proposed architectural model are [16] :

- MANET interfaces are seen only by a router, assumed to be MANET aware and running appropriate protocols and applications;
- MANET interfaces forming a multi-hop MANET area may use a site (not subnet) prefix (aggregation,...);
- hosts/subnets on non-MANET interfaces assume a classic IP link model;
- applications on hosts see classic IP interfaces connected to a classic IP link, and therefore;
- applications on hosts and protocols assuming classic IP interfaces can run unmodified.

2.5.2 Scope equivalences

In order to regard MANETs as an equivalence of wired LANs, it is especially important in the field of autoconfiguration to have a clear definition of different scopes. These scopes are directly derived from the IP architecture defined in RFC 3513 [40]. Note that in the following only IPv6 addressing will be considered but the concepts apply as well for IPv4 (refer to RFC 1518 [71]).

Local scope

With local scope or link-local scope in classical IP addressing all nodes on the same link are considered (see figure 2.6). All hosts on the same link sharing the same prefix (i.e. being in the same subnet) can directly communicate in the link layer of the ISO/OSI

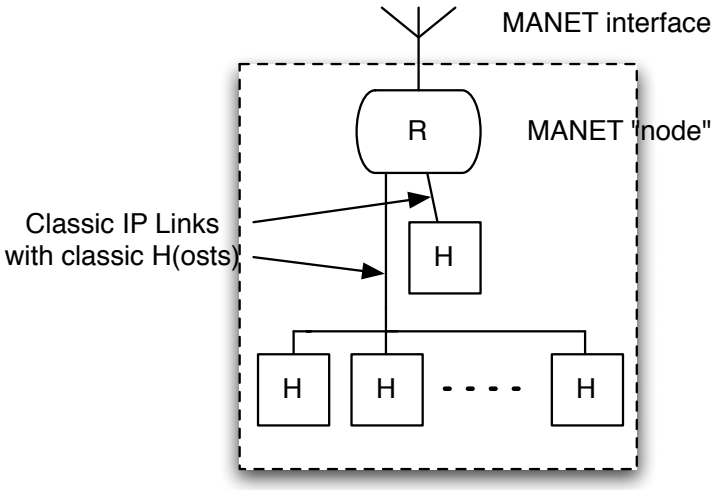


Figure 2.9: MANET node corresponding to the proposed architecture

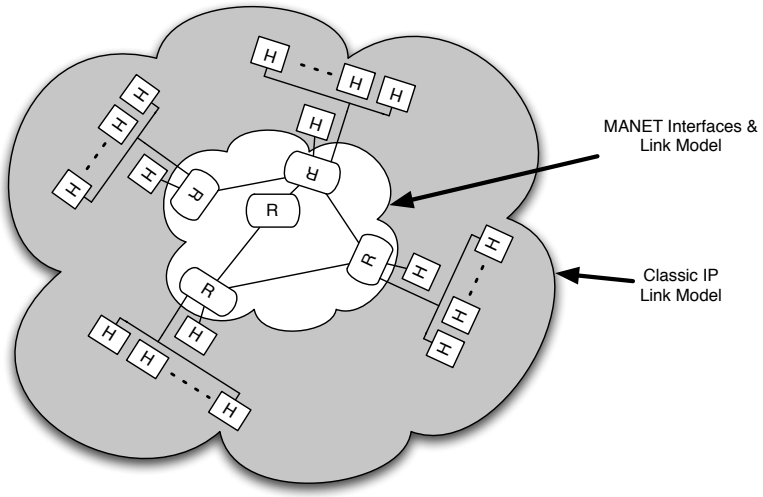


Figure 2.10: MANET model

model. As explained above, a local scope includes all hosts in one MANET node. It also includes direct 1-hop-neighbors of a node as they can communicate directly without routing. In the special case of OLSR as a routing protocol, also 2-hop-neighbors count to a local scope. Unique addresses in a local scope means that for any two nodes A and B which are 1-hop neighbors, A and B have different addresses.

Site scope

The next larger scope than a local one would be a site-local scope. While the definition for the link-local scope is rather obvious, a definition of a site is not canonical. In fact, site local addresses are declared deprecated in RFC 3879 [44]. There are several issues with site-local addresses:

- What is actually a site? Is it a company's network? Or is it limited to a geographic location? How to handle intermittently connected networks, mobile nodes, mobile networks inter-domain VPNs, hosted networks, network merges

and splits? [44]

- Site-local addresses are ambiguous. Applications have to deal with hosts coming from different sites but having the same prefix. Nodes do not know whether a node with a site-local prefix belongs to its own site or another connected site.
- Even if site-local addresses should never appear in the Internet, these addresses might appear globally by passing through leaks in web pages or files. For example, they might be used in TCP packets as source address, for DNS or trace-route requests. Also reverse DNS queries may cause lots of unnecessary traffic.

For these reasons, one should no longer use these addresses. However, as some of the autoconfiguration algorithms are based on the use of site-local addresses, they are defined in this thesis as addresses for all nodes that belong to one MANET. In this sense, a MANET consists of mobile nodes belonging all to the same connected graph via wireless links running the same routing protocol. As the notion of site-local is deprecated, this concept will henceforth be called *MANET-local*. As mentioned in section 2.5.1, using a common MANET-local prefix for all nodes in a MANET is a misperception of the addressing concept. Unique addresses in a MANET-local scope are unique within one MANET, but may be ambiguous in a global scope.

Global scope

When one talks about a global scope in wired but also in wireless networks, this means the entire aggregation of all networks to the Internet. All these networks are connected via routers to each other and packets may be routed from any host to any other (without considering firewalls, NATs, etc.). In order to communicate this way, all routers have to have globally unique prefixes. So a unique address in a global scope is unique within the whole Internet and may not be ambiguous to any other address in this connected graph.

2.6 Prefix aggregation

In this section the address format for nodes is derived from the architectural model of section 2.5.1 and the scopes described in section 2.5.2. According to this architectural view, each node is its own subnet. Thus, using subnets for aggregation (CIDR [42]) is not possible any more. Instead, using a common prefix part for the purpose of aggregation is proposed.

In the example of figure 2.11, several MANETs `p`, `p`, `r` are connected among them using aggregation [42]. MANET `q` has a connection to an Internet gateway. When using a common prefix part `d` for all MANETs in this "site", this gateway can use aggregation for all MANETs. As a consequence, prefixes for routers would be of the form: `d:p:s::/64`, where `d` is a site-local prefix part, `p` the prefix part for the MANET and `s` the unique prefix part for a router.

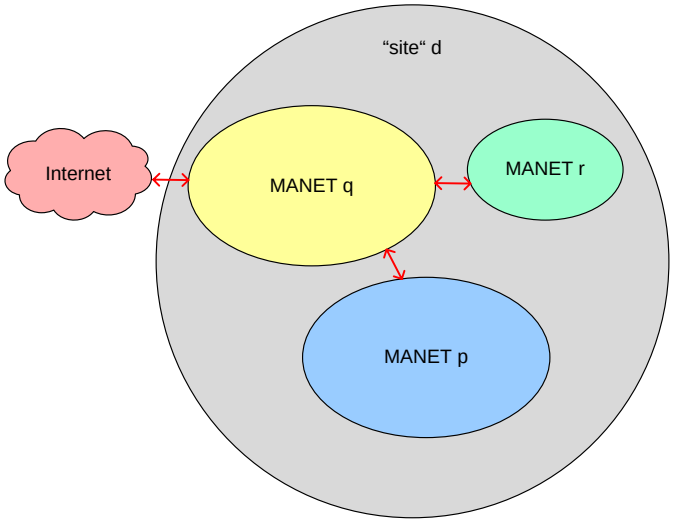


Figure 2.11: Introducing a common site local prefix d for aggregation

3 Autoconfiguration classification and terminology

In order to be able to compare the different autoconfiguration approaches in a structured manner, a classification is necessary. In this chapter, such a classification is specified and existing protocols are allocated to this classification. However, only some known protocols are presented in this thesis; for a more detailed overview of protocols refer to [9, 47]. The second part of this chapter will consist of a terminology section. As many of the autoconfiguration papers use different words for the same meaning – or even worse – the same term for different meanings, it makes sense to define the interpretation of terms for MANETS in this thesis. Throughout the overview of related works in chapter 4, the defined terminology of this chapter will be used instead of the papers' exact wording in order to stay consistent in the survey.

3.1 Classification

The proposed classification is depicted in figure 3.1. The issue of autoconfiguration is structured as follows: On the uppermost level the "ultimate" goal of autoconfiguration is

> *to configure globally unique and topologically correct IP addresses*.

Note that the goal only includes configuration of IP addresses while autoconfiguration could also mean automatic attribution of standard gateways, DNS servers etc. This would also be possible but is out of the scope this thesis.

As it is not always possible or necessary to have globally unique addresses, this goal is divided into several subgoals dealing with different regional scopes:

- **Configuration of locally unique addresses**
  This means that only in the 1- or 2-hop neighborhood addresses are assured to be unique. This is a subgoal which is necessary for most autoconfiguration algorithms if any two adjacent nodes want to communicate.
- **Configuration of MANET-local unique addresses**
  These are addresses that are unique within one standalone MANET (see section 3.2 for a definition). This requirement assures communication, including routing, within a standalone MANET not being connected to the Internet.
- **Configuration of Internet gateways and distribution of global prefixes**
  The attribution of globally unique prefixes to nodes of a MANET enables them to connect to the Internet and allow packets to be globally routed.

Below the level of subgoals there are several possible solutions or approaches to achieve these subgoals. They are explained in the following section 3.2. Papers proposing autoconfiguration algorithms may use one or several of these approaches to achieve one or several of the subgoals.

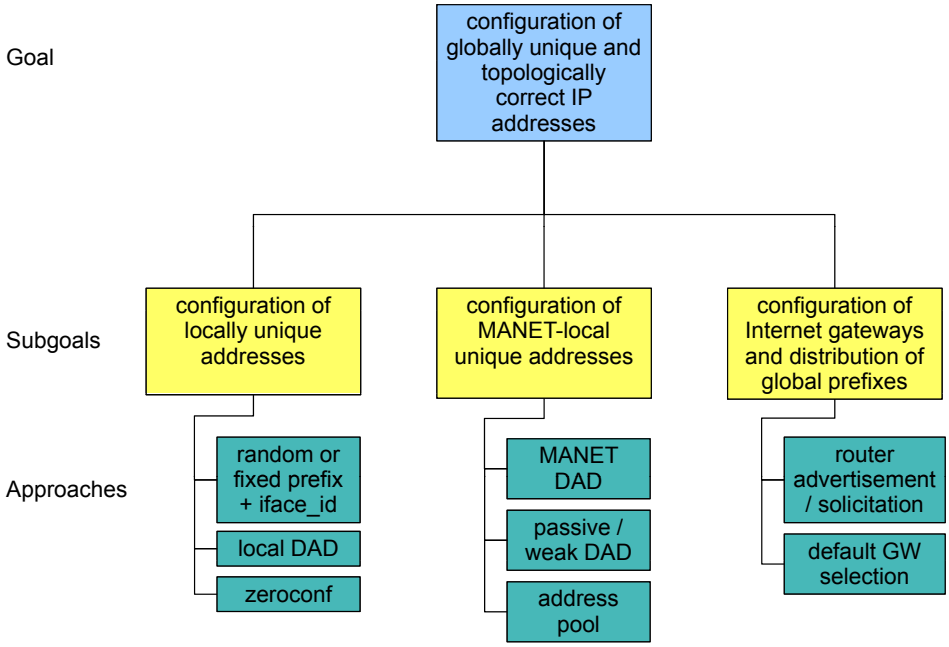


Figure 3.1: Autoconfiguration algorithms classification

3.2 Terminology

This terminology is based on [5] and tries to enlarge the extent of [5] by adding a classification of terminology. This may lead to terms not used by other papers and also contradict some of the terms used by these papers as a different model of MANETs is considered in this thesis. The following three categories are used to discern different kinds of terms: entities, processes and properties.

3.2.1 Entities

An **entity** is something that has a distinct, separate existence, though it need not be a material existence [87]. In MANETs, all participating objects (like nodes, routers, hosts, gateways) are entities which may interact with each other.

- MANET
  A mobile ad hoc network (MANET) is a connected directed graph $G = (V, E)$ where V are mobile nodes and E are wireless connections between the mobile nodes. These connections may be unidirectional or bidirectional. A MANET may be connected to the Internet or other wired or wireless networks via one or several gateways. See figure 2.10 for the architectural model of a MANET applied in this thesis.
- mobile node
  A mobile node consists of a router having a router prefix and one or several attached hosts (see section 2.5.1 for a more detailed description). Mobile nodes can communicate to each other adjacent node using their wireless interface operating in ad hoc mode. Nodes route packets for other nodes which are not in a 1-hop neighborhood by using a common MANET routing protocol.
- router
  A router is a role of a mobile node. Its primary task is to route packets from any

host of the MANET to any other using a predefined routing protocol.

- host
  A host is an optional role of a mobile node. It could either be an application within to node or a physically or virtually attached host to the node. A host gets the router prefix assigned and needs a host ID.
- site
  The term "site" is deprecated (refer to RFC 3879 [44]). In the wireless case, a site could be one or several adjacent MANETs as defined above.
- link
  A link in a MANET is a bidirectional connection between two adjacent nodes (see figure 2.6). All nodes on a link should be able to directly communicate without routing. A precise definition of a link is difficult, however, and subject of many discussions in the IETF.
- standalone MANET
  A standalone MANET is a MANET which is not connected to any other network by a gateway. For autoconfiguration this means that the MANET does not necessarily have a globally unique address.
- Internet gateway
  An Internet gateway is a mobile node which has a connection (either wireless or wired) to the Internet. Thus, mobile nodes in the MANET are permitted to send packets to hosts in the Internet. An Internet gateway has to advertise itself (either proactively or passively) in order to get known by the mobile nodes and to distribute its globally unique prefix.
- globally connected MANET
  In contrast to the standalone MANET, a globally connected MANET is reachable from the Internet by an Internet gateway. This requires all nodes of the MANET to have a globally unique prefix.

3.2.2 Processes

A **process** (lat. processus - movement) is a naturally occurring or designed sequence of changes of properties/attributes of a system/object. More precisely, and from the most general systemic perspective, every process is representable as a particular trajectory (or part thereof) in a system's phase space [88]. Processes can be subdivided into different groups: events and operations. Note that in this subsection only terms are listed that are not directly related to autoconfiguration approaches. This means that terms like duplicate address detection, active/passive or weak/strong duplicate address detection are not included in this terminology. This group of terms (which could be called methods) will rather be explained in the related works part.

Events

An event in a MANET is something observable that may occur at any given time. It includes network topology changes like network merging and partitioning.

- network merger
  A network merger happens when two formerly distinct MANETs approach so that they have a wireless connection between at least one node of either MANET. As the mobile nodes of two MANETs may have used the same IP addresses, the network merger has to be somehow detected and duplicate addresses have to be prevented.

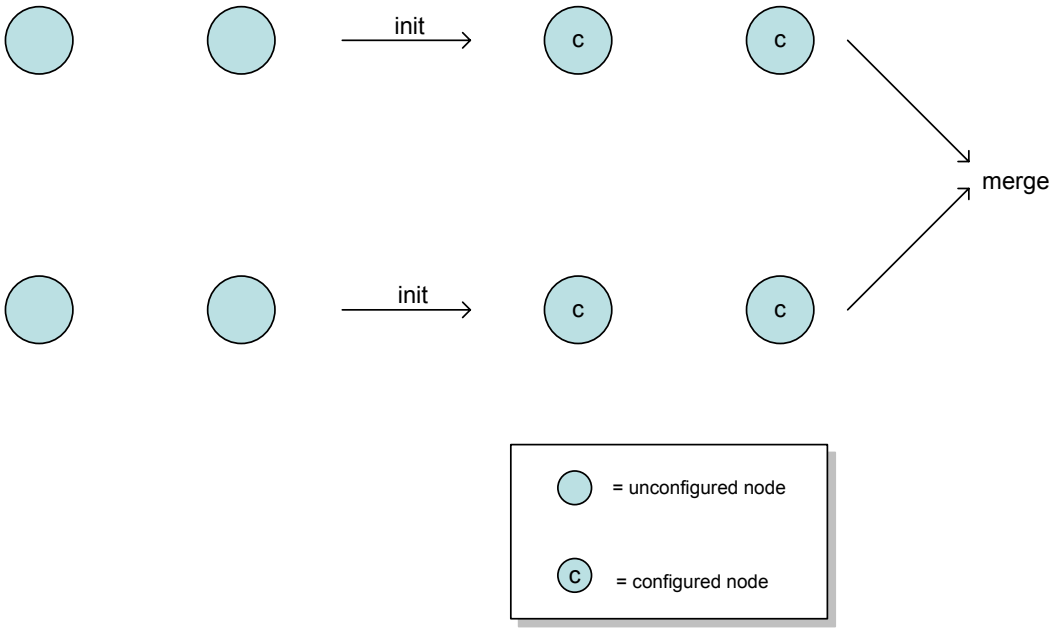


Figure 3.2: Nodes being initialized first an then merge in a next step

- network partitioning
  In a network partitioning, a MANET gets divided into at least two parts. This may happen if two formerly connected components of a MANET move too far away from each other.
- node join
  This event occurs whenever a new node wants to join a MANET. An initialization event is triggered to configure the node correctly.
- node initialization
  When a node joins a MANET, it has to get configured. Most importantly, it has to get an IP address in order to be able to communicate with the other nodes of the MANET. See figure 3.2 for an example where first nodes get initialized and then merge.
- node leave
  This event occurs when a node leaves a MANET. This can either happen in a planned way or abruptly (e.g. if the node crashes).

Operations

An operation is an instruction that may be triggered by an event. For example in the case of a node initialization event, a node may send a broadcast with a tentative IP address.

- MANET broadcast
  When a mobile nodes wants to broadcast a packet, it wants this packet to be delivered to all nodes in the MANET, transmitting it only once. Nodes have to forward the packet to their neighbors. Nodes may receive the same packet several times due to the nature of wireless connections (wireless interfaces communicate on the same communication channel and incoming and outgoing packets are sent over the same channel). If this is the case, they have to discard the duplicated packet in order to avoid infinite flooding with one packet.

- classic flooding
  Classic flooding is the simplest way of broadcasting a packet. Nodes forward a packet to all or their neighbors exactly once. This is the most secure way of broadcasting, as it is assured that all connected nodes will receive the packet (without considering packet loss). However, many duplicate packets are sent around in the MANET causing a high congestion rate.
- MPR flooding
  Multipoint relay (MPR) flooding is a more effective way of broadcasting packets through the MANET when using OLSR as routing protocol. A node applying MPR broadcasting selects only part of its neighbors for forwarding. This results in a more optimized broadcasting mechanisms (refer to section 2.4.1 for more details about MPR flooding).

3.2.3 Properties

A property of an object is some intrinsic or extrinsic quality of that object [89]. An object in this sense would be one of the named entities of section 3.2.1.

- IP address
  IP addresses are part of the IP protocol. Their intention is to uniquely address a host for communication purposes. IP addresses consist of a network prefix and a host identifier. They are either 4 bytes long (in IPv4) or 16 bytes in IPv6. More information can be found in RFC 1518 [71] and RFC 3513 [40] respectively.
- prefix
  A prefix is part of an IP address and is defined in RFC 3513 [40] and RFC 1518 [71] respectively. A prefix is generally a certain number of bits from the beginning of an IP address defining a range of nodes.
- site local address
  An IP address having the site-local prefixes 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16 or `FEC0::/10`.
- local address
  An IP address having a link-local prefix 169.254/16 or `FEB0::/10`.
- global address
  A globally valid and unique IP address.

4 Related work on autoconfiguration algorithms

As stated in the chapter 2, autoconfiguration algorithms are an important part of research in the field of MANETs. However, much more work has been done in routing protocols, and no autoconfiguration protocols have been standardized. In 2005, the Autoconf Workgroup of the IETF has been created to gather work done in this field and to standardize protocols.

In this chapter, the state of the art of autoconfiguration algorithms is presented and classified in the structure presented in section 3.1. This survey is not intended to be complete but rather to present the most known approaches. A more complete overview can be found in in [9].

This chapter will be divided into two parts: First of all, autoconfiguration for classical wired IP-based networks are presented and then a selection of the most important and recent approaches for MANETs will be described.

4.1 Classical wired IP networks

In a classical IP network (as depicted in figure 2.6), there are several used techniques to configure IP addresses to all hosts of which the most known will be presented in this section.

4.1.1 Bootstrap

One of the first protocols assigning a unique IP address to a host is Bootstrap (BOOTP) [30] which was standardized in 1985[1]. BOOTP is a IP/UDP protocol originally designed to boot diskless clients which for example needed to access a boot image of a remotely hosted Unix before booting this image. Necessarily, the client needed to have an IP address to communicate with this remote server and in addition information about the file location and name. In a next step the client would then download the image using a protocol like TFTP.

BOOTP consists of a single exchange of messages: the client sends a bootrequest packet and the server responds with a bootreply packet. If the client does not know the address of the server, it will use a broadcast address (255.255.255.255) as destination. The source address is 0.0.0.0 (i.e. the invalid address). The server then looks up the MAC address of the client and a corresponding IP address in a table and sends it back (for example using the broadcast address again).

There is an extension to forwarding requests and responses over gateways if the bootstrap server is not on the same link as the client.

The protocol is less used than the newer DHCP as it offers less possibilities and DHCP is backward compatible. However, the protocol is much simpler than DHCP.

1 There were earlier protocols like RARP which were less convenient as they were not based on IP but directly on the hardware level

4.1.2 DHCP

The Dynamic Host Configuration Protocol (DHCP) [32] assigns automatically IP address, net mask, standard gateway etc. to a node using a central server – a DHCP server. DHCP is backward compatible to BOOTP, so any BOOTP client can collaborate with a DHCP server without any adaption. This description focuses on IPv4, however, there is a standard for DHCPv6 as well (RFC 3315 [33]).

There are three modes in DHCP:

- **manual allocation**
  The DHCP server has a table with MAC addresses and corresponding IP addresses. Only clients in the list can be assigned the predefined IP address which has to be manually configured in the table by a system administrator.
- **automatic allocation**
  The DHCP allocates to any requesting client a free IP address from a predefined range of IP addresses. This address belongs permanently to the client. When there is no more IP address left in the pool, new clients cannot be allocated an IP address. The only possibility is to reset the IP address pool and delete all registered allocations.
- **dynamic allocation**
  Clients request and get granted an IP address for a certain period of time. This concept is called lease.

Whenever a clients enables its network interface and wants to acquire an IP address, it has to find a DHCP server first. Therefore, it broadcasts an DHCPDISCOVER message to the broadcast address 255.255.255.255 which includes its MAC address. Note that there may be several DHCP servers on the same link, so they all can receive the DHCPDISCOVER message. The servers then respond via broadcasting with a DHCPOFFER message offering possible IP addresses for the client. The client has to decide for one offer (e.g. by choosing the first incoming one) and tell all servers about his choice by broadcasting a DHCPREQUEST message including the server identifier from which the address proposal has been chosen. Finally, the server acknowledges the requested address by sending a DHCPACK message to the client including information like the lease time and other additional information. The server might also deny the request of the address by sending a DHCPNAK message. The simplest standard process is depicted in figure 4.1.

4.1.3 Zeroconf

Zeroconf has been developed by Cheshire et al. from Apple when they moved from AppleTalk to IP as network protocol. It has been standardized in RFC 3927 [15]. So far, Zeroconf addresses three issues of which only the first will be explained in this section.

- IP address allocation
- DNS name resolution
- Service discovery

Whenever a node needs to be configured a valid link-local IP address, it picks a random address in the reserved range of 169.254.1.0 to 169.254.254.255. The pseudo-random number generator should be implemented such that different nodes do not choose the same random number sequences. In particular, they should not be seeded with the local time as they may have the same time and thus choosing the same number sequences. Nodes could be seeded with for example their MAC address. This also eases the process

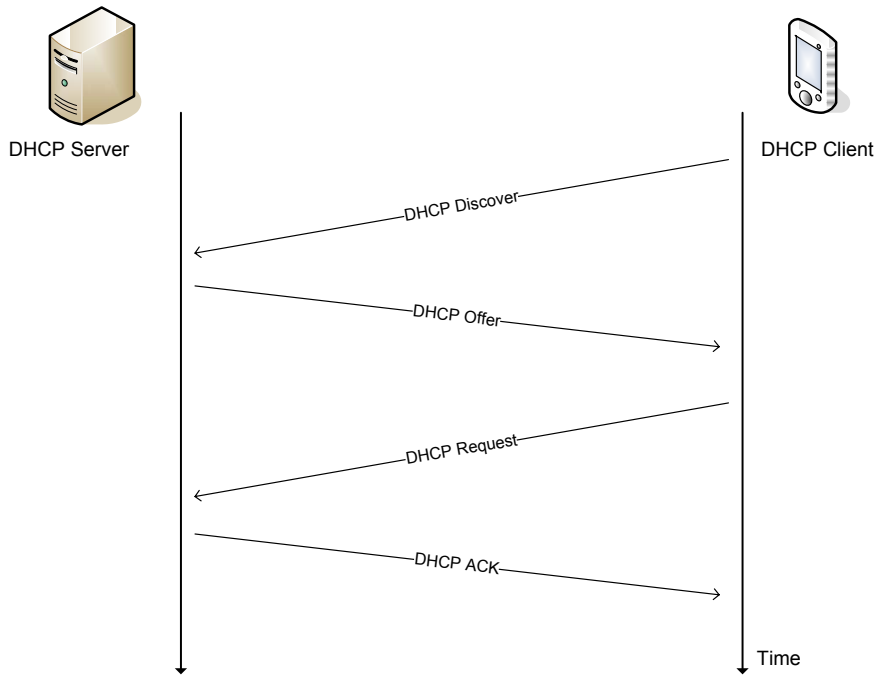


Figure 4.1: Simple DHCP communication between a client and a server

when an interface gets disabled and enabled later again. When it uses a constant seed, it will get the same IP address when being re-enabled and conflicts get avoided.

However, when a node has picked the tentative address, it should perform a uniqueness tests of the address. This is performed using ARP messages. The configurating node broadcasts an ARP request with the tentative IP address as target IP address, 0.0.0.0 as source IP address (to avoid cache congestion of other nodes), and its own MAC address as hardware source. After a random amount of time, the node sends a certain predefined number of probes to the link.

If the nodes receives either an ARP reply or request with the tentative IP address as source address or target address respectively, it should take the next random number from the sequence. Receiving an ARP reply means that another node has already picked this address while receiving an ARP request with a different MAC source address means that another node wants to configure the same address.

Note that the uniqueness test is performed only once. However, checks for ARP replies and requests are constantly performed by all nodes on the link. Whenever a node receives an ARP message with the sender IP address being the address of one of its interfaces but the target hardware address differs from the hardware identifier of the node, there has to be an address conflict. The node should then interrupt any connections to or from it and choose a new address.

Note that link-local addresses are for local use only (see section 2.5.2 and [71]). They should never be forwarded by a router. Zeroconf has been implemented in Microsoft Windows and Mac OS in 1998.

4.1.4 IPv6 stateless autoconfiguration (IPv6 SA)

IPv6 stateless autoconfiguration (IPv6 SA) is defined in RFC 2461 [62] and RFC 2462 [76]. The design goal of IPv6 stateless autoconfiguration was to minimize manual user interaction. Hosts and router IP addresses should be configured automatically and no ad-

ditional servers (for example DHCP) should be necessary. Hosts create their addresses from their host identification (e.g. MAC address) attached to the link-local prefix in IPv6. After having verified the uniqueness of this address by using a duplicate address detection, they can complete their configuration by looking for site-local or global prefixes from routers. The advertisement of these prefixes is accomplished by the routers. Note that [76] does not include router prefix autoconfiguration. For more information about router configuration refer to section 4.1.5.

In the following a list of the steps in IPv6 SA is presented:

1. **Creation of a tentative link-local address**
   This occurs whenever a network interface has been enabled or attached to the link. The interface uses the well-known link-local prefix `FE80::0` [40] (of appropriate length) and attaches its interface MAC address to the prefix (using [4]).
2. **Duplicate address detection**
   The node has to verify that its link-local address is unique on the link. Therefore it broadcasts Neighborhood Solicitation (NS) messages using the neighbor discovery protocol [62]. If the node receives a Neighbor Advertisement (NA) response from any node on the same link, it knows that its tentative address is already allocated to another node. In this case, the node has either to use a new interface identifier, or the autoconfiguration process fails and another method of assigning an IP address has to be applied.
3. **Assignment of the link-local address**
   If the duplicate address detection test passes successfully, the node may finally assign the link-local address as permanent. The node can now communicate with all other nodes on the same link but not with other nodes behind a router (e.g. a node in the Internet) because packets with link-local addresses as destination or source may never be routed.
4. **Getting a prefix from a router**
   Assuming there is at least one router attached to the link, a node may receive the prefix from the router. Therefore it can either send a Router Solicitation (RS) message or wait for a periodically broadcasted Router Advertisement (RA) packet. This message contains the prefix of the router.
5. **Assigning a site-local or global prefix**
   Under the assumption that stateless autoconfiguration is used in the network, the node may create its new site-local or global address by simply attaching its interface ID to the received prefix. Another duplicate address detection is generally not necessary as in the test before the interface identifier has been proofed as being unique.

4.1.5 Zerouter

Another interesting approach is Zerouter which is similar to Zeroconf but focuses on autoconfiguration of *routers*. There has been done some research in 2002 and 2003 [14, 31, 38, 55, 56, 57, 64, 85, 86], however, none of the drafts have been published. Design goal and motivation of Zerouter was to enable home users and small companies without competencies in network administration to build arbitrarily complex and large networks without manual interaction. Especially routers should be able to completely autoconfigure themselves which includes in particular the configuration of prefixes. As in the architectural model of MANETs, mobile nodes are considered as routers with attached hosts (refer to section 2.5), these proposals are interesting for autoconfiguration in MANETs.

There are different approaches to configure routers with Zerouter. A common proposal however is that there is one delegating router which is chosen among routers at one link. This router has the task to delegate a prefix to its link. In order to perform this task, it has to know about other adjacent routers in the site. Thus, it gets to know the site-wide prefix as Top-Level Aggregation Identifier (TLA)[2]. In addition, the delegating router has to choose a Site-Level Aggregation Identifier (SLA) and assure its uniqueness within the site. Hosts attached to routers in Zerouter have to be configured by other means (e.g. by using IPv6 stateless autoconfiguration as described in section 4.1.4).

4.2 MANET autoconfiguration

As explained in section 3.1, there are three subgoals of the primary goal to configure globally unique and topologically correct IP addresses for all nodes. Current papers propose approaches which aim to achieve one or more of the subgoals. In the following, these approaches will be presented classified by the three subgoals.

4.2.1 Configuration of locally unique IP addresses

In order to communicate with adjacent nodes from the MANET, a node upon initialization may pick a link-local address. As in a MANET each node is a router, a node having a link-local address may only communicate with its direct neighbors.

Tentative address with uniqueness test

A common approach of papers is the one introduced by [69]: A node first picks a temporary IP address from the range 1-2047 of the class B network 169.254/16 (or from `FE80::/64` for IPv6). These addresses should never be assigned as a permanent address for a MANET node. Then the node attaches either part of its MAC address or the whole MAC address to the prefix. In IPv6 networks the node has first to convert the 48 bit MAC address into a EUI-64 address (refer to [4] for more details). Another method would be to attach a random number to the link-local prefix. This may be useful for anonymity purposes for example.

As these addresses may be duplicate even in the 1- or 2-hop neighborhood, it is advisable to perform a duplicate address detection.

Unique local addresses (ULA)

Another possibility is presented by [45] which is inspired by Unique Local IPv6 Unicast addresses [41]. The idea is to take `FEB0::/10` as a prefix and choosing the following 54 bits randomly. Then again, either the MAC address could be attached or the last 64 bits can be chosen randomly as well (as depicted in figure 4.2). In this approach, flat addressing is assumed, i.e. each node has a 128-bit prefix (for IPv6) or a /32 prefix (for IPv4). Both approaches use a global scope. However, the prefixes are considered to be routed only within a site in [41] and a MANET in [45] respectively.

The probability of a collision of two ULA addresses (when nodes have the same MAC address) can be estimated using the general birthday-paradox with n being the number of nodes that may collide and $d = 2^{54}$ being the number of possible random addresses to choose from:

(4.1) $p(n;d) \approx 1 - e^{-(n(n-1))/2d}$

2 Refer to RFC 2374 [42] for more information about aggregatable unicast addresses in IPv6

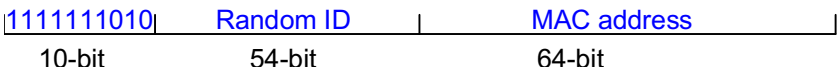


Figure 4.2: Unique Link-Local Addresses

Unnumbered interfaces

Furthermore, there exist so-called unnumbered interfaces as specified in RFC 1812 [6]. Usually, routers in IP networks have a prefix and also their own IP address. [6] however, discusses the use of unnumbered lines and network prefixes. According to the author, a router does not necessarily have to have an IP address, as it is never itself part of an end-to-end communication line. However, this causes some issues with the classic IP architecture. Particularly, the next-hop IP address is used for routing which causes problems when using unnumbered routers. The author suggests using two routers which operate as "half-routers". Together, they act like a normal router. Problems arising from this constellation are that the communication between the two half-routers is not standardized and may be different between routers of different manufacturers. In addition, if several routers are used in a mesh network, behavior of routers may be difficult to set up. As this is so far the case in MANETs, no proposal has so far tried to implement unnumbered routers for local communication.

4.2.2 Configuration of MANET-local unique IP addresses

Nodes wishing to communicate mutually in a MANET need to have unique address within a MANET-wide scope. If any two (or more) addresses within a MANET are duplicated, the routing protocol is not able to distinguish any more between the conflicting nodes and packets will be delivered to the wrong node or will be dropped. There are several approaches to assure MANET-local unique addresses.

Active DAD

MANET-DAD One of the first drafts proposing a typical active DAD algorithm is [69] and has been extended by more recent papers. A node having picked a temporary address floods the MANET by sending the tentative address in an *Address Request (AREQ)* to all adjacent nodes. A node receiving an AREQ first creates a unicast route back to the node. It then compares the tentative address from the originating node with its own. If they do not match, the node forwards the AREQ packet to all of its neighbors. If the addresses match (i.e. the tentative address of the node performing autoconfiguration has already been assigned by the receiving node), the node sends back an *Address Reply (AREP)* using the unicast reverse route.

The autoconfigurating node waits a certain time for receiving an Address Reply message after having sent the request. If it receives a reply within the time limit, it has either to choose another tentative address or has to be configured manually. If no reply has been received however, the node resends its request for a certain predefined maximum number of tries. Still not having received a reply after these tries, the node assumes that no node in the MANET has chosen the same address and keeps the IP address permanently.

IPv6 stateless autoconfiguration (IPv6 SA) adopted for MANETs [83] adopts IPv6 stateless autoconfiguration for wired networks [76] to mobile ad hoc networks. As in [76], neighbor solicitation (NS) messages are used to detect if the chosen tentative

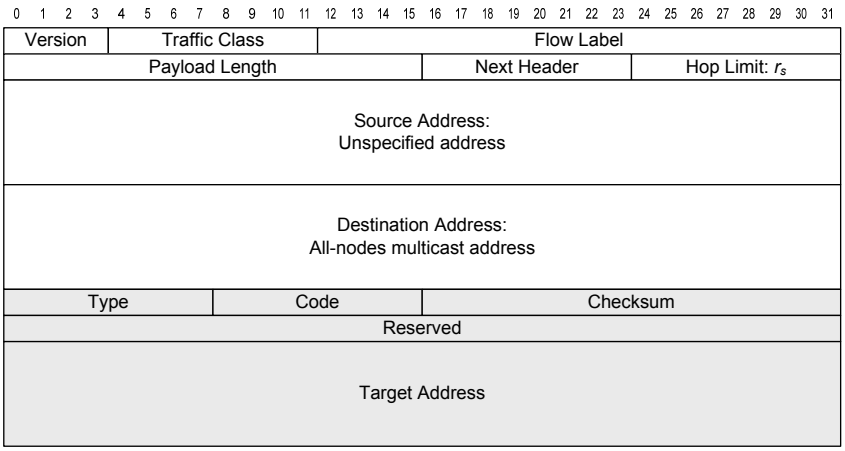


Figure 4.3: Neighborhood solicitation message format

link-local address has already been assigned. Figure 4.3 depicts a neighbor solicitation message for IPv6 SA in wired networks. The NA message uses the unspecified address as source address and includes the tentative address in the message body. Compared to classical IPv6 SA however, [83] uses the all-nodes multicast address as destination address and includes a hop limit of r_s. Thus a NS packet is broadcasted from the originating node to all its neighbors. A node receiving a NS message decreases the hop count by one and forwards it if $hopcount > 0$. Thus, address uniqueness is guaranteed within the scope (limited by r_s) of each node. In order to discern NS messages from different nodes performing DAD and having the same tentative address, a new option called *MANET option* is introduced in the NS messages. This option contains a *Random Source ID (RS-ID)* field with a random value enabling the distinction of such two nodes.

Proxying In addition to the proposed methods, there exits some extensions as presented in [48] or [73]. Whenever a AREQ or NA message respectively arrives at a node, it not only compares the tentative address with its own address, but also with addresses in its routing table and its cache. This can reduce the overhead and congestion in a MANET on a large term, since broadcasting is often limited to a smaller scope.

Routing-protocol specific protocols

There are several protocols which depend on specific properties of the routing protocol. For instance, several protocols ([1, 10, 18, 17, 74, 75]) are proposed to be used with OLSR. As in this thesis the objective of an autoconfiguration algorithm will be to be independent of the routing protocol, these protocols are not presented here.

Weak and passive duplicate address detection

Weak DAD This mechanism is proposed by [79]. The author assumes that even after a duplicate address detection like the one proposed by [69], there may be duplicate addresses within a MANET. Therefore, he proposes a so-called weak duplicate address detection which constantly tries to find duplicate addresses even after the initial configuration process. Nodes include a pair (IP address, key) in their packets. The key has to be unique and could be the MAC address for example. Whenever a node receives

packets from one IP address but with different keys, it knows that there has to be a conflict. This algorithm depends on certain attributes of the routing protocol as it has to somehow include the pair (address, key).

[48] combines the approaches of weak DAD [79] and strong DAD [69] in one algorithm. It supports both IPv4 and IPv6 and is not restricted to any particular routing protocol.

Passive DAD Passive autoconfiguration algorithms try to defer from a certain number of predefined rules whether there is a conflict. The idea is to avoid overhead by sending packets throughout the MANET to detect an address conflict but rather to listen the network in a passive way. A design goal of such algorithms is obviously less overhead and congestion in the MANET while still detecting all conflicts. The problem is, however, to assure detecting all these conflicts. In addition, there often have to be many rules for detecting a conflict, inflicting complicated and time-consuming calculations on each node.

Typical passive DAD algorithms are [81] and [82].

4.3 Conclusion

There exist many different proposals of how to configure IP addresses for MANETs. However, they all base on the architectural misperception presented in section 2.5.1. This may be one of the main reasons, why none of the protocols have been standardized. As a consequence, in chapter 5 an autoconfiguration algorithm is presented which is integrateable into the current IP infrastructure.

5 Proposal of an autoconfiguration algorithm

There exist many solutions for autoconfiguration of MANETs which have been shown in chapter 4, and classified in chapter 3. However, all of these have been designed under the "misperceived" architectural view shown in section 2.5.1. As presented in the related works part in chapter 4, these algorithms pose several problems:

- They do not correspond to the classical IP model presented in section 2.5.1, thus:
- Nodes in a MANET have a common subnet prefix which would imply they could directly send messages on the link-level without routing.
- The proposals are often only partial and thus not addressing all subgoals of autoconfiguration presented in section 3.1.
- Proposals often have lots of assumptions like the underlying routing protocol in the MANET.

In this chapter, an autoconfiguration algorithm will be developed which overcomes these problems and is oriented towards the classification presented in section 3.1. In addition, the proposed protocol will correspond to the architectural view presented in section 2.5.1 – this is, it will consider MANET nodes as routers with physically or logically attached hosts.

At first, the motivation and design goals for the autoconfiguration algorithm will be specified in section 5.1 and an overview of the different steps of the algorithm will be given in section 5.2. Then the algorithm will be developed from section 5.3 on, beginning with the most basic version accomplishing autoconfiguration of mobile nodes but not yet all design goals of section 5.1. In the following sections, extensions and optimizations will be developed to fulfill all design goals.

5.1 Motivation and design goals

The autoconfiguration algorithm of this thesis is constructed with the following design goals:

- It should correspond to the architectural model presented in section 2.5.1 that MANET nodes are routers with potentially attached hosts and therefore:
- MANET nodes need to have unique prefixes within the MANET.
- The prefixes of MANET routers should be easily aggregatable using Classless Inter-Domain Routing (CIDR) [71].
- Hosts should be automatically configured using the prefix of the router they are connected to.
- All subgoals of the classification depicted in figure 3.1 should be accomplishable: autoconfiguration of locally unique addresses, autoconfiguration of MANET-local unique addresses and configuration of Internet gateways and distribution of global prefixes.
- The achievement of one subgoal should not depend on the accomplishment of another (e.g. allocation of MANET-local prefixes should not depend on link-local addresses).

- The algorithm should be applicable in IPv4 and IPv6.
- Network overhead should be reasonable low.
- The solution should be independent of any particular routing protocol.

5.2 Background and overview

The idea of the algorithm proposed in this thesis is based on two algorithms: Zerouter (refer to section 4.1.5) and IPv6 stateless autoconfiguration (refer to section 4.1.4). Note that only IPv6 is considered (refer also to section 6.4). As presented in the architectural model in section 2.5.1, MANET nodes are routers with zero or more hosts attached. Thus, the autoconfiguration problem represents the problem of how to autoconfigure *routers*, not hosts. The Zerouter approaches have exactly the same goal of autoconfiguring routers but are assuming wired LAN connections. As one design goal presented in section 5.1 was to allow prefixes in IP addresses to be aggregated using CIDR [42], it is proposed to use a common prefix part in all IP addresses of a MANET. Several solution presented in chapter 4 already suggest this. However, they consider this prefix part to be a subnet (i.e. being 64 bit long in IPv6). This contradicts the architectural model of a MANET presented in section 2.5.1 because this would imply that all MANET nodes could directly communicate within the MANET. So the idea of the autoconfiguration algorithm in this thesis is to use a prefix for routers and hosts which consists of three parts: A first part `d` for the whole site, then the second part `p` which is common for all nodes in a MANET and allows aggregation using CIDR [42]. The third part `s` however is assumed to be unique for each router. As `p` is shorter than 64 bits and `s` supposed to be different for all MANET nodes, no MANET router is part of the same subnet and thus the architectural model presented in section 2.5.1 is not violated. On the other hand, aggregation of prefixes in routing tables is still possible.

In order to achieve all of the subgoals of section 3.1, the algorithm is divided into several steps which are depicted in figure 5.1 and will be overviewed in the following:

1. **MANET routers acquire link-local addresses**
   In order to perform the exchange messages on a link-local scope (refer to section 2.5.2 for a discussion of scopes in a MANET), a node should have a link-local IP address. Confirming to the design goals of section 5.1 and as described in section 5.4.1, it will be shown that this is however not a precondition for autoconfiguring MANET-local addresses. Nevertheless, it may reduce ambiguities of message distribution when using unique link-local addresses. If link-local addresses are desired, several solutions are presented in section 5.3.
2. **The *router* role of each MANET node has be configured with a MANET-local prefix**
   The router first tries to get the MANET-wide prefix part `d:p` by one of his 1-hop neighbors (henceforth called initiator node). It then chooses a prefix `d:p:s::` consisting of the acquired parts `d` and `p` from the initiator node and a unique random part `s`. It verifies the uniqueness of `s` by sending it to the initiator node which broadcasts it throughout the MANET. Any node having the same prefix part `s` sends back a warning to the initiator node which returns it to the configurating node. If there is no duplicated prefix, the initiator node will send an acknowledgment to the configurating node. The algorithm is presented in section 5.4.
3. **IPv6 stateless autoconfiguration (IPv6 SA) configures *hosts* attached to the routers**
   After a router has acquired its prefix, attached hosts have to be configured using IPv6 SA [76]. Hosts attached to the router create an IP address by attaching their

EUI-64 MAC address [4] to the link-local prefix. They verify the uniqueness of the MAC address by broadcasting Neighborhood Solicitation messages on the link. If no conflict occurs, they get to know the router prefix by broadcasting Router Solicitation (RS) messages on the link. The algorithm is presented in section 5.5.

4. **Internet gateways distribute their prefixes and register MANET nodes**
   In order to enable Internet access, border gateways to the Internet have to distribute their global prefixes and mobile nodes have to be configured correctly. Refer to section 5.6.

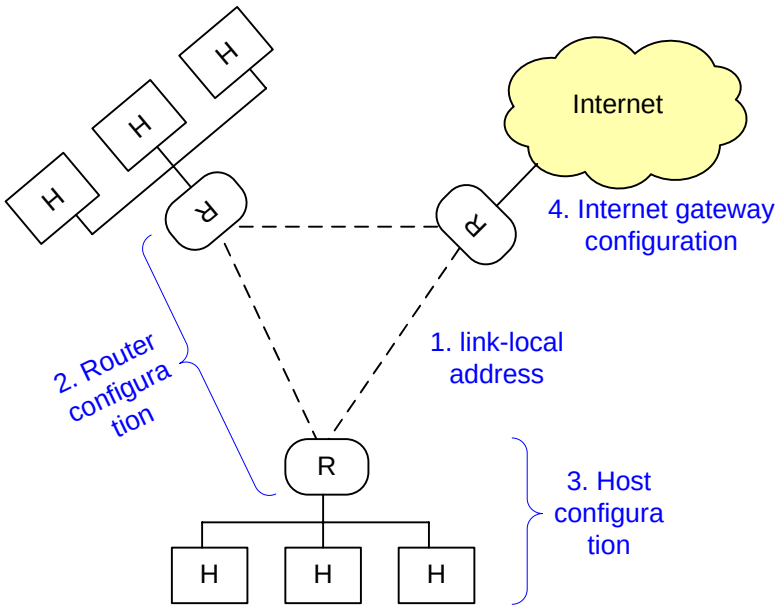


Figure 5.1: Autoconfiguration steps in the proposed autoconfiguration algorithm

5.3 Assignment of a link-local address

As classified in section 3.1, one subgoal of any autoconfiguration algorithm is to have unique link-local addresses. Whenever a node wants to exchange messages with its direct 1-hop neighbors it needs some way of determining the target destination.

Note that the link-local addresses only have to be unique in a 2-hop neighborhood (see figure 5.2 for an example). In this example, four nodes have link-local addresses and can thus communicate in their 1-hop neighborhood (note that link-local addresses may never be routed by a node – refer to [40]). The arrows in the figure depict the communication range (e.g. node A may communicate with node B but not with node C; node B can reach both nodes A and C). As link-local addresses are never routed, nodes A and D may have the same link-local addresses without any conflict. However, if nodes A and C have the same addresses, messages sent from node B would arrive ambiguously at node A **and** C.

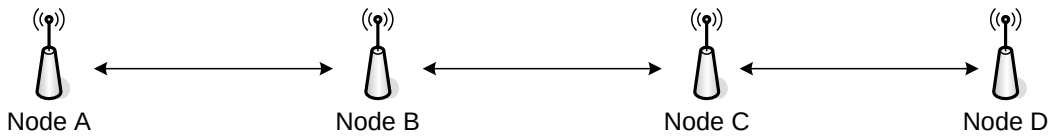


Figure 5.2: Link-local neighborhood

Table 5.1: Approaches for acquiring a link-local address

Approach	Duplicate address detection	IP version
Zeroconf	2-hop test	IPv4
Unique Link-Local Addresses (ULLA)	not required	IPv6
IPv6 neighborhood discovery	2-hop test	IPv6
Unnumbered interfaces	not required	IPv4 and IPv6

There are several approaches for how to assign link-local addresses (refer to table 5.1 for an overview of the proposed approaches). The autoconfiguration algorithm of this thesis may use any of them. They mainly differ in the necessity of a duplicate address detection in the 2-hop neighborhood and the required IP version. Note that the autoconfiguration of MANET-local addresses in section 5.4 is not affected whether the nodes have link-local addresses at all or how they acquired them. It also does not matter whether they are unique or duplicated (refer to section 5.4.1). In the following sections, four approaches for acquiring link-local addresses that are unique in a 2-hop scope are presented.

5.3.1 Unique link-local addresses (ULLA)

The solution is based on Unique Local Addresses (ULA) [41] and MANET local addresses (MLA) [45] as described in section 4.2.1. In these two solutions, however, the prefixes are considered to be routed only within a site in [41] and a MANET in [45] respectively. Thus, none of both approaches is directly applicable for link-local addresses in MANETs as their scope is too large. Site-local addresses would be routable among nodes which is not desired for link-local addresses. In addition, site-local addresses are defined as deprecated in [44] as there is no clear definition of a site (refer also to section 2.5.2).

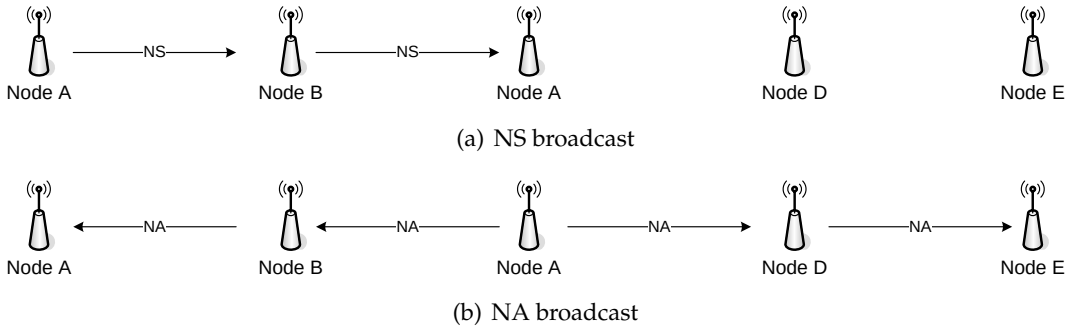
The proposed mechanism for creating link-local addresses (henceforth denominated as Unique Link-Local Addresses (ULLA)) is using the same approach as in [45] but using the well-known 10-bit link-local prefix `FE80::/10` instead of `FC00::/8`. A ULLA address thus consists of the 10-bit link-local prefix, and a 118-bit long random ID.

Using a ULLA address as link-local address assures a high probability of uniqueness as the addresses have only to be unique in the 2-hop neighborhood (as depicted in figure 5.2). The probability of a collision can be estimated using the general birthday-paradox as explained in section 4.2.1. With a 118-bit random number and assuming a large number of for example 10000 nodes in the 2-hop neighborhood, the probability would be no more than $1.5 \cdot 10^{-28}$. The probability of duplicate addresses in the 2-hop neighborhood is considered as low enough to abstain from duplicate address detection mechanisms. In addition, the proposed autoconfiguration algorithm for MANET-local addresses does not depend on unique link-local addresses (refer to section 5.4.1). However, to assure ULLA addresses to be unique in a 2-hop scope beyond the very small probability, one can additionally perform a duplicate address detection similar to the one presented in section 5.3.2.

5.3.2 IPv6 neighborhood discovery

Another possibility for configuring link-local addresses would be to use an approach based on IPv6 stateless autoconfiguration (IPv6 SA) [62, 76] (refer to section 4.1.4 for more details). For this algorithm to work in the MANET in order to create unique link-local addresses among 2-hops, the algorithm has to be changed.

As NS and NA messages in classical wired IP networks are considered to operate only on one link (i.e. may not be routed), one has to find a way to broadcast the messages up to 2-hops away. In the stage of configuration of link-local addresses one cannot assume any kind of existing routing protocol, so a simple broadcast mechanism is proposed in the following. The explanation is based on the example depicted in figure 5.3.



(a) NS broadcast

(b) NA broadcast

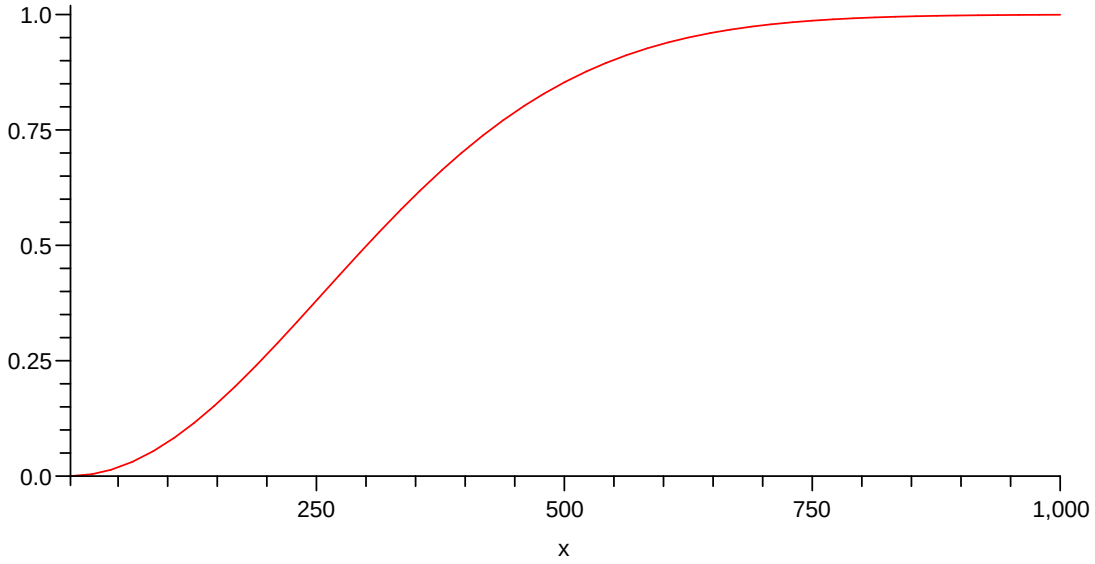
Figure 5.3: 2-hop neighborhood discovery for duplicate link-local address detection

1. A node chooses a tentative link-local address by attaching the EUI-64 [4] MAC address to the link-local prefix.
2. It then broadcasts a Neighborhood Solicitation (NS) message containing the desired link-local address in the message body (refer to [62] for the packet format of NS and NA packets). In order to achieve only a 2-hop broadcast, the hop-limit of the packet is set to 2 and the hop-count to 0.
3. Whenever a node receives a NS message, it first increments the hop count. It then compares the desired tentative address from the NS message with its own address. If they are equal, it broadcasts a NA message with a hop-count of 2. Otherwise, it either forwards the NS message if the hop-count is smaller than the hop-limit or otherwise discards it.
4. Whenever a node receives a NA message containing its tentative address, it chooses a new address by for example creating an ULLA address (refer to section 5.3.1). If the address from the NA message is different, it either forwards it and increments the hop-count, or it discards it if the hop-count is equal to the hop-limit.
5. If the node performing the uniqueness test does not receive a Neighborhood Advertisement (NA) message within a timeout `RetransTimer`, it retries sending an NS message. This is repeated `DupAddrDetectTransmits` times (as defined in [76]).
6. If still no NA message containing the tentative address has been received, the link-local address is chosen as permanent.

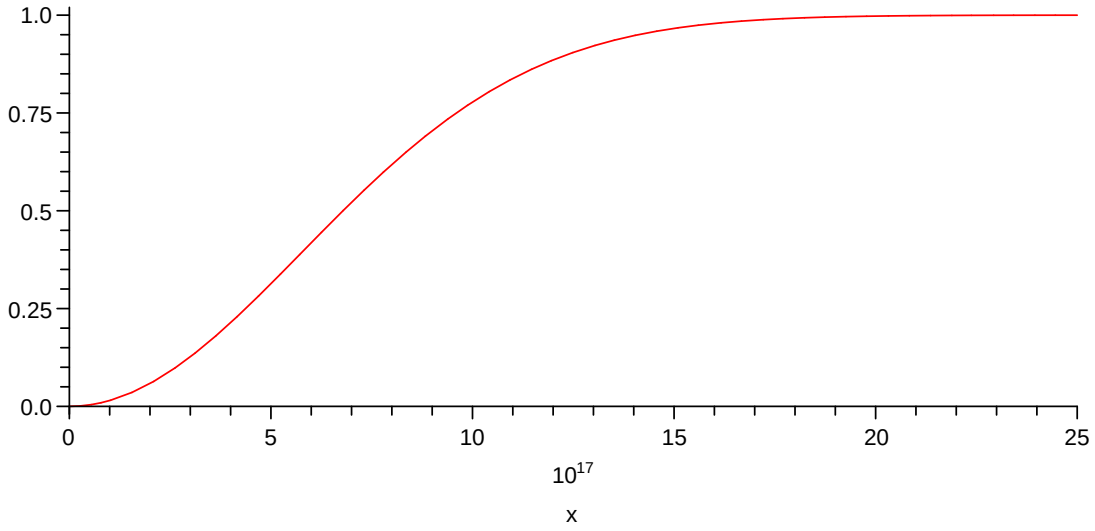
5.3.3 Zeroconf

Zeroconf [15] is the equivalent of IPv6 stateless autoconfiguration [76] for IPv4 addresses (refer to section 4.1.3 for a more detailed description). As claimed in section 5.1, the autoconfiguration algorithm should also be applicable for IPv4 addresses (even if the this thesis focuses on IPv6 – refer also to section 6.4). When using IPv4 addresses with the proposed autoconfiguration algorithm, Zeroconf may be used to configure link-local addresses. Zeroconf has to be adapted for MANETs in order to being able to send ARP messages to a 2-hop range. Notice that the collision probability of two nodes having the same link-local address within a 2-hop range when using IPv4 addresses is

substantially higher than using IPv6 and one of the proposed algorithms of section 5.3.1 or 5.3.2 (see figure 5.4 for a probability distribution). Having for instance 500 nodes in a 2-hop neighborhood produces a collision probability of 85% in IPv4, while using IPv6 and ULLA addresses the collision probability with the same number of nodes is as low as $3.8 \cdot 10^{-31}$.



(a) IPv4 with 65024 possible addresses



(b) IPv6 with 2^{118} possible addresses

Figure 5.4: Collision probability of x nodes in the 2-hop neighborhood when using Zeroconf (IPv4) and 118-bit ULLA addresses (IPv6) random addresses respectively

5.3.4 Unnumbered interfaces

As presented in section 4.2.1, unnumbered interfaces allow to configure routers without allocating IP addresses to them. In most cases, routers will only directly communicate with each other to configure their prefixes and routes. Only hosts are considered to exchange messages between them as end-to-end points. Using unnumbered interfaces in the proposed autoconfiguration algorithm of this thesis to configure MANET-local addresses would be sufficient as the algorithm does not depend on link-local addresses.

Nodes only have to be able to send and receive link-local broadcasts. However, by using unnumbered interfaces the subgoal of assigning unique link-local addresses would not be achieved.

5.3.5 Conclusion for assigning link-local addresses

In the previous subsections different methods have been shown to provide a unique link-local address within a 2-hop range which is sufficient for link-local communication. The choice of the used algorithm depends on which IP version is used and whether link-local addresses are needed at all, how much network overhead is tolerated, and whether an (even very small) probability of having duplicate addresses is acceptable or if addresses have to be unique with 100% certainty.

Whatever the choice for acquiring a link-local address, it does however not influence the proposed algorithm of allocating MANET-local addresses in section 5.4. As explained in section 5.3, for a link-local communication only a 2-hop uniqueness is required. However, if one wants to assure unique link-local addresses within a larger scope, one may easily use one of the approaches of either section section 5.3.2 or section 5.3.3 and replace the hop-limit of 2 by the desired range.

If link-local uniqueness is only guaranteed within a 2-hop scope and a frequent communication using link-local addresses is desired, perhaps one has to constantly repeat the uniqueness test. For example, as depicted in figure 5.2, four nodes have configured 2-hop unique link-local addresses. As link-local addresses are not routed, it does not matter that two nodes A and D have chosen the same address `fe80::1`, as long as they do not approach. If however node D moves into the direct vicinity of either node A or B, communication gets ambiguous. A larger scope of the uniqueness test may help to avoid these problems but increases the network overhead.

5.4 Router configuration

In this section the autoconfiguration protocol for routers is presented. As one design goal of the algorithm was to be independent of unique link-local addresses, another way of discerning nodes has to be used. In the following subsection 5.4.1 a means of identifying nodes uniquely is proposed before in section 5.4.2 the protocol itself is explained.

5.4.1 UUIDs

As standardized by the Open Software Foundation (OSF), the concept of Universally Unique Identifiers (UUID) is considered in order to uniquely discern nodes. These 16-byte long numbers can identify nodes even when not using link-local addresses. In addition, they can be easily implemented as there are several standard libraries coming with operating systems.

Each node creates a UUID at its initialization which is henceforth included in the local message exchange.

By using the concept of UUIDs, link-local addresses can be completely avoided. Note that UUIDs are only included in local exchange of messages and are not comprised compulsorily in the following MANET broadcast. Thus a large spread of UUIDs in the MANET may be prevented. For a discussion what happens when UUIDs overlap, refer to section 6.6.1.

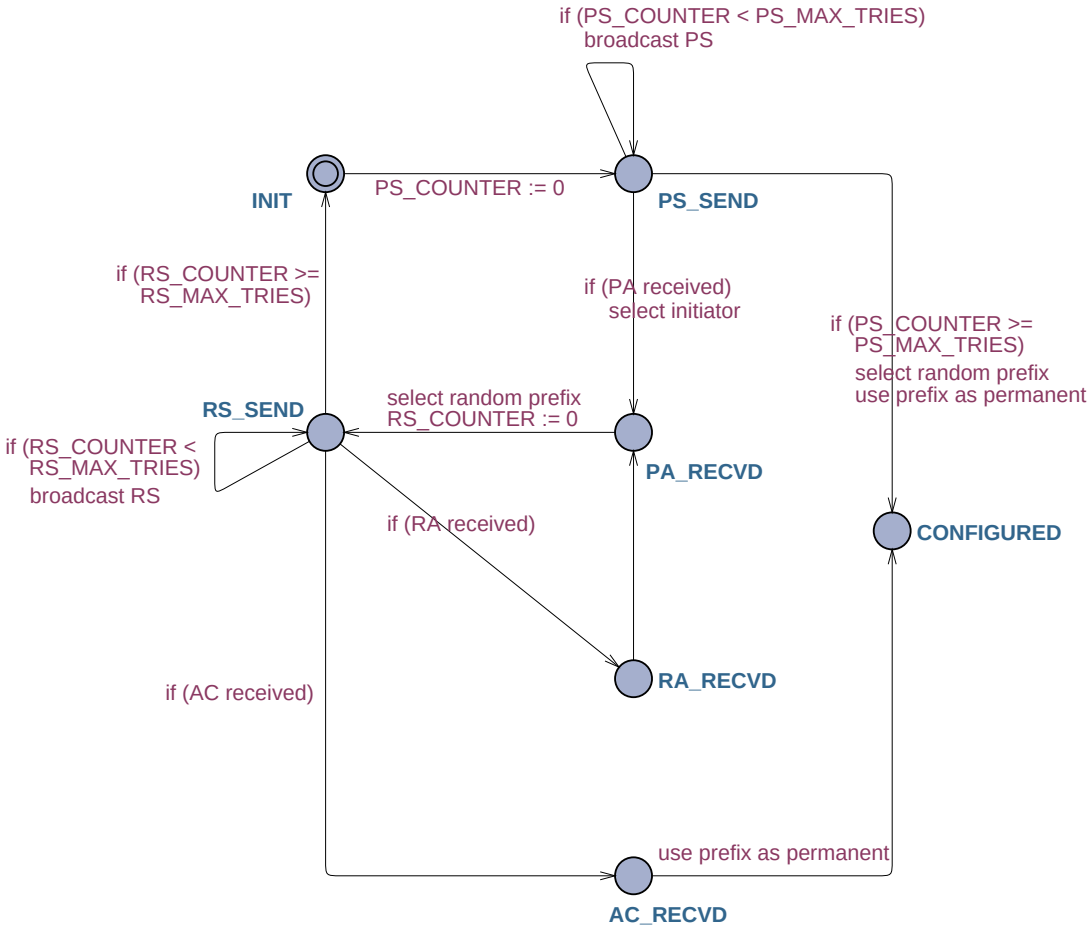


Figure 5.5: State machine of the algorithm for configuring MANET-unique prefixes

5.4.2 Proposed protocol

As stated in the introduction (section 5.2), the next step after configuring link-local addresses is to configure routers. In the following subsections, the proposed algorithm for configuring MANET-local prefixes is developed.

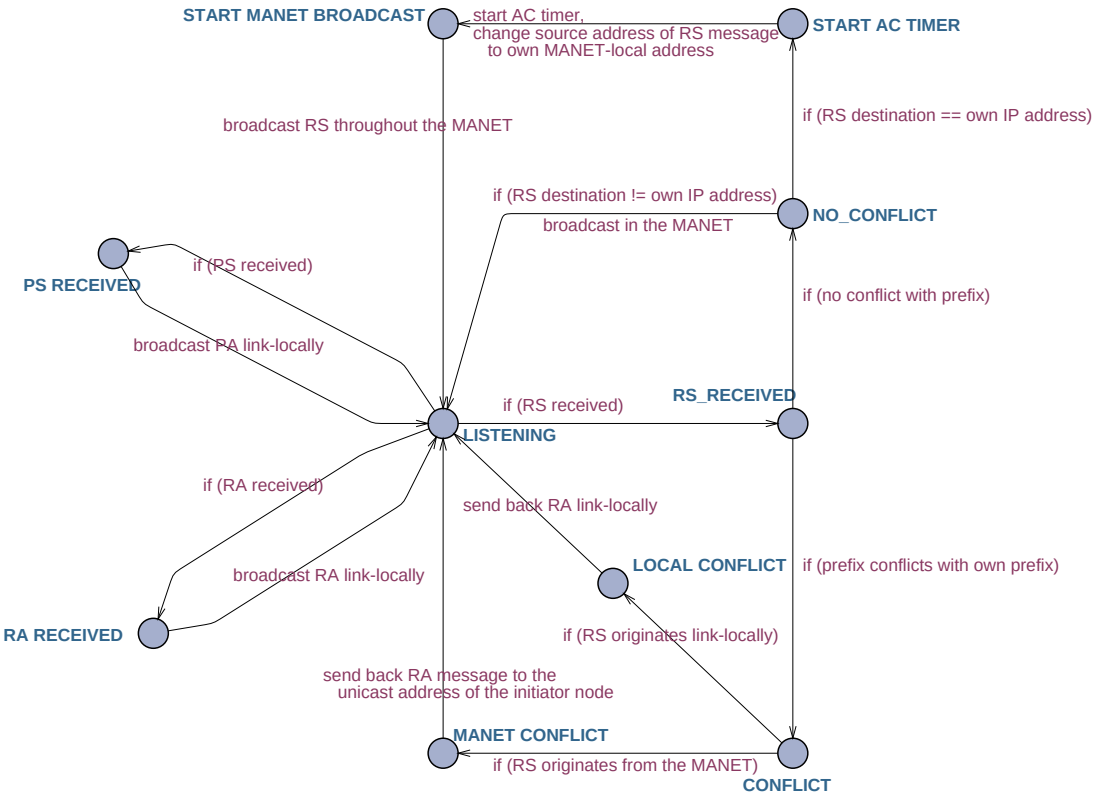
A node that wants to communicate in a MANET has to choose a router prefix and must assure that it is unique within the MANET-scope. As one of the design goals of the algorithms is to facilitate aggregation of prefixes, the prefix attributed to each node will be of the form `d:p:s::/64` with `d` being the site prefix part, `p` being the MANET prefix part and `s` being the MANET router part (as described in section 2.6). Note that the site prefix part `d` may be of length 0. A MANET node has to perform two steps:

- It has to acquire the first parts `d` and `p` of the prefix.
- It has to assure that the second part `s` is unique within the MANET.

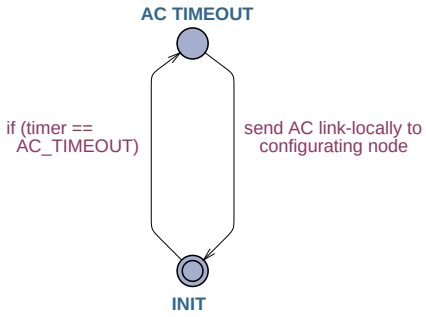
The algorithm which is developed in this section complies to this two-step approach. The exchanged messages base on the IPv6 stateless autoconfiguration algorithm [62, 76] but use the MANET generalized packet/message format [21] for its extensibility.

In figure 5.5 the steps of a node to become autoconfigured are depicted, and in figure 5.6 the steps for an already configured node handling an autoconfiguration request of a new node are displayed both as state machines.

1. **The node broadcasts a Prefix Solicitation (PS) message**
   This step corresponds to the transition from `INIT` to `PS_SEND` in figure 5.5. The



(a) State machine for initiator nodes



(b) State machine for AC timer

Figure 5.6: State machine of the algorithm for configuring MANET-unique prefixes

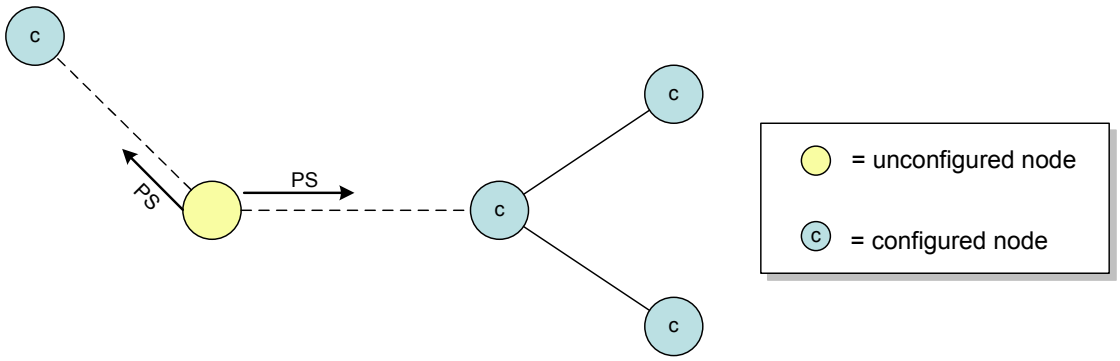


Figure 5.7: A configurating node broadcasts a Prefix Solicitation (PS)

node broadcasts a PS message to all of its 1-hop neighbors in order to get the common prefix parts `d` and `p` from one of them (depicted also in figure 5.7). Refer to A.1 for details about the contents and structure of the used messages.

2. **Two possible cases**

   a) **No answer has been received**
   When no answer is received within a certain timeout `PS_TIMEOUT`, the node tries again sending the PS message. This process is repeated at maximum `PS_MAX_TRIES` times (depicted by the transition from `PS_SEND` to `PS_SEND`). If then still no answer has been received, the node assumes that it is the first node of the MANET. It then uses a predefined site-local prefix part `d`, creates a random MANET prefix part `p` of length `PREFIX_LENGTH` using a pseudo-random number generator and concatenates a random router prefix part `s`. This corresponds to the transition to `CONFIGURED`. The algorithm terminates.

   b) **A Prefix Advertisement (PA) message has been received**
   If one or more direct neighbors of the configurating node receive a PS message, they may reply per broadcast with a Prefix Advertisement (PA) message transmitting their prefix (in particular the first prefix parts `d` and `p`). In addition, the PA message includes the UUID of the sending node.

3. **The configurating node selects one PA message that has been received**
   For example, it selects the first message that has been received[1] (transition `PS_SEND` –> `PA_RECVD`). It extracts the prefix information from the message. Now the node knows the prefix parts `d` and `p`. The node from which the PA message has been sent is called *initiator node* from now on as it will assist the configurating node in the prefix uniqueness test later on. The configurating node stores the UUID of the initiator node which was included in the PA message.

4. **Next the node chooses a random prefix part `s`**
   This new prefix `d:p:s::/64` is only a tentative prefix as it might be duplicated within the MANET. No messages with this prefix may be sent unless the uniqueness of the prefix has been acknowledged. This step is depicted in the transition from `PA_RECVD` to `RS_SEND`.

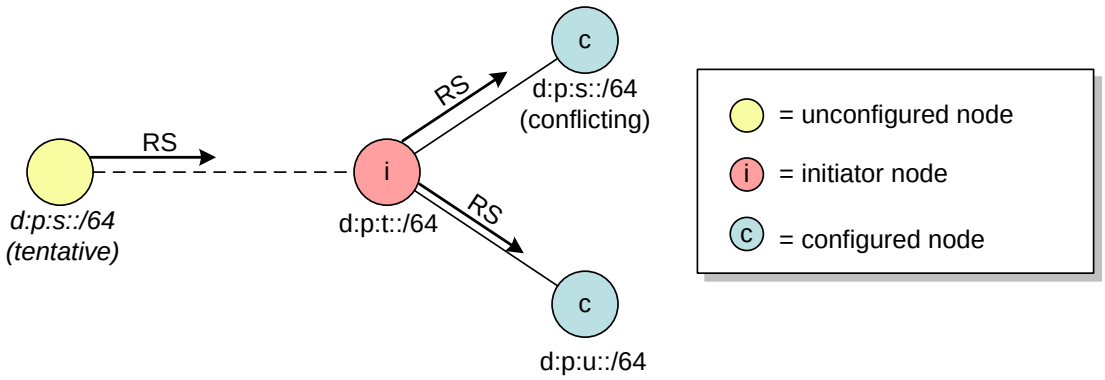
5. **The node sends a Router Solicitation (RS) message to the initiator node**
   Corresponding to the transition from `RS_SEND` to `RS_SEND`, the RS message contains the new tentative prefix `d:p:s::/64`. The source address of the configurating node is the unspecified address and the destination address is the link-local broadcast address. In addition, the UUID of the configurating node as well as the target UUID of the initiator node are included in the RS message (refer

1 Refer to section 6.5 for a discussion about the initiator selection

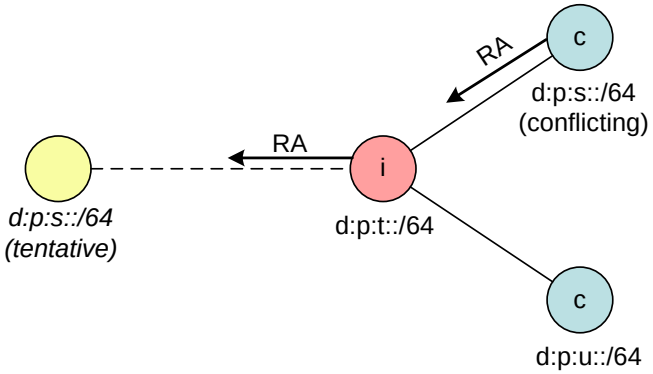
to figure 5.8 for an example). The RS message is sent a maximum number of RS_MAX_TRIES times.

6. **The initiator node receives the RS message and broadcasts it to the MANET after having changed the source and destination address**
   The initiator node listens for RS messages with its own UUID as target destination (depicted in figure 5.6(a), LISTENING state). It first checks whether the tentative prefix d:p:s::/64 is identical to its own prefix. If this is the case (state CONFLICT), it sends back a Router Advertisement (RA) message to the configurating node (state LOCAL_CONFLICT). If it is different to its own prefix, it changes to source address of the RS message to its own MANET-local IP address. After that it broadcasts this message to the MANET.
7. **Two possible cases**
   a) **A node in the MANET receives the forwarded RS message and has the same prefix as the tentative one from the message**
   If that is the case, the node with the conflicting address sends back a Router Advertisement (RA) message with its own prefix using the same broadcasting mechanism as used for the RS message (transition MANET_CONFLICT –> LISTENING). The algorithm continues at step number 8a.
   b) **A node in the MANET receives the RS message and has a different prefix as the tentative one from the message**
   As there is no conflict, the node broadcasts the RS message to all of its neighbors (transition NO_CONFLICT –> LISTENING). The algorithm continues at step number 8b.
8. **Two possible cases**
   a) **If the initiator node has received a RA message due to a prefix conflict, it forwards it to the configurating node**
   The initiator node changes the source address to its own link-local address and broadcasts the packet link-locally (transition RA_RECEIVED –> LISTENING). All configurating nodes receiving that RA message with the tentative prefix included in the RA message have to change their tentative prefixes by choosing a new random prefix part s and continuing at step number 5 (transition RA_RECVD –> PA_RECVD, figure 5.5).
   b) **If the initiator node has not received any RA messages which are addressed to itself, it resends the RS message**
   In order to assure uniqueness of the prefix, the initiator node retries sending the RS message after a timeout RS_TIMEOUT. This is repeated until a RA message is received or RS_MAX_TRIES tries have already been accomplished. If still no RA message has been received, the algorithm continues at step number 9.
9. **The initiator node sends an Autoconfiguration Confirmation (AC) message to the configurating node**
   After the initiator node has waited AC_TIMEOUT seconds without receiving any RA answer, it sends an Autoconfiguration Confirmation (AC) message to the configurating node including the UUID of the configurating node (as depicted in figure 5.6(b). As soon as the configurating node receives this AC message, it accepts its tentative prefix as permanent (transition AC_RECVD –> CONFIGURED). Now the node has to configure the host IP addresses using the router prefix (refer to section 5.5).

For an example of a typical autoconfiguration process in a time diagram see figure 5.9. In the upper part, 5.9(a) displays an autoconfiguration example without any conflict



(a) The initiator node forwards a RS message from the configurating node via broadcast



(b) A conflicting node sends a RA message back to the configurating node

Figure 5.8: RS broadcast example

while 5.9(b) depicts an autoconfiguration conflict. In figure 5.9(a), at first the configurating node $config_1$ sends its PS messages. After several tries it receives a PA answer from the node called $init$. After two tries of sending a RS message, $init$ receives this message and broadcasts it throughout the MANET. As no conflict has occured (i.e. no RA messag has been receives), the $init$ sends an AC message to $config_1$. Figure 5.9(b) basically displays the same message exchange, only that a conflict with a node in the MANET occurs (i.e. $init$ receives a RA message). This RA message is then forwarded to $config_1$.

As will be shown in the chapters 7, 8 and 9, the proposed algorithm configures correctly unique MANET-local prefixes without being depend on a routing protocol or any link-local addresses. However, the performance can be optimized by using techniques which are presented in chapter 6.

5.5 Host configuration

After a node has acquired a unique prefix by completing the steps from section 5.4, it has to configure its hosts. Remind that a mobile node is considered as a router with zero or more hosts attached to it (refer to section 2.5.1). These hosts can either be logical (i.e. only a role integrated into the mobile node) or externally attached hosts (i.e. by another network interface).

Depending on the used IP version, different methods of configuring the host address are proposed:

5.5.1 Host configuration with IPv6 stateless autoconfiguration

The proposed configuration method of these hosts is using IPv6 stateless autoconfiguration [76]. For a detailed description of IPv6 stateless autoconfiguration refer to section 4.1.4.

As soon as all hosts have been configured using IPv6 SA, they can communicate with any other host in the MANET or in any other connected network using classical routing (refer to [40]).

5.5.2 Host configuration for IPv4 hosts

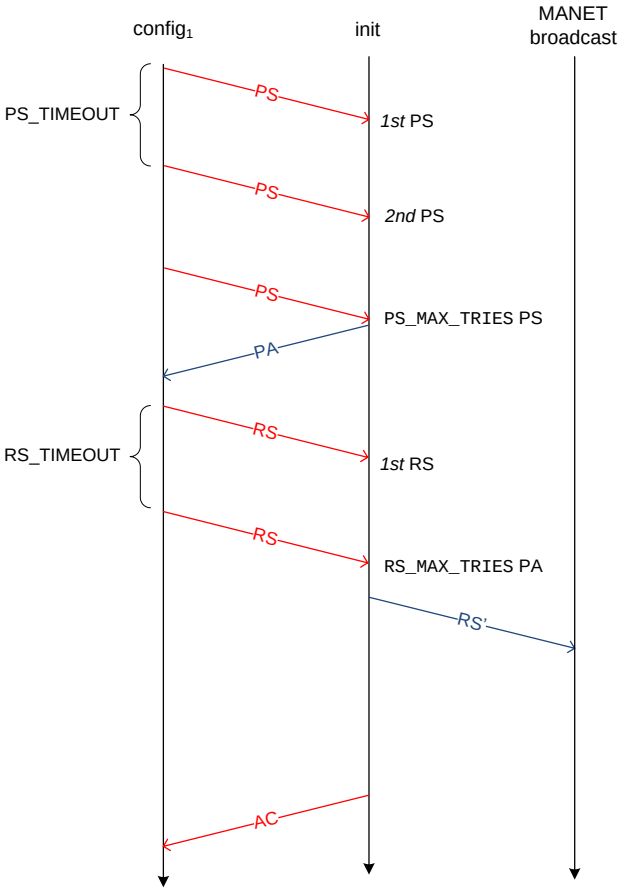
As the focus of this thesis is IPv6, no detailed description of host configuration with IPv4 will be proposed. However, Zeroconf [15] could be used to configure host addresses. A detailed description of Zeroconf can be found in section 4.1.3.

5.6 Internet gateway configuration

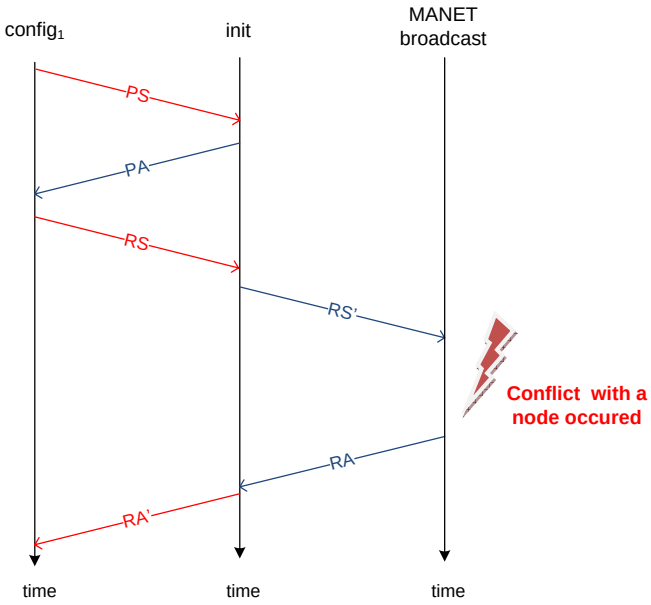
After having configured MANET-local prefixes to the routers and IP addresses to the hosts as described in sections 5.4.2 and 5.5, Internet gateways – if available – can be configured and global prefixes can be attributed to the nodes. There are several proposals for this (e.g. [43, 46, 54, 58, 66, 73, 74, 83]), however, this topic will not be treated in this thesis.

5.7 Partitioning and Merging

MANET mergers and partitioning (refer to section 3.2.2) impose certain problems on autoconfiguration. As a detailed discussion was not possible in the limited time of this



(a) Autoconfiguration example without conflict



(b) Autoconfiguration example with conflict

Figure 5.9: Autoconfiguration proceeding example as time diagram

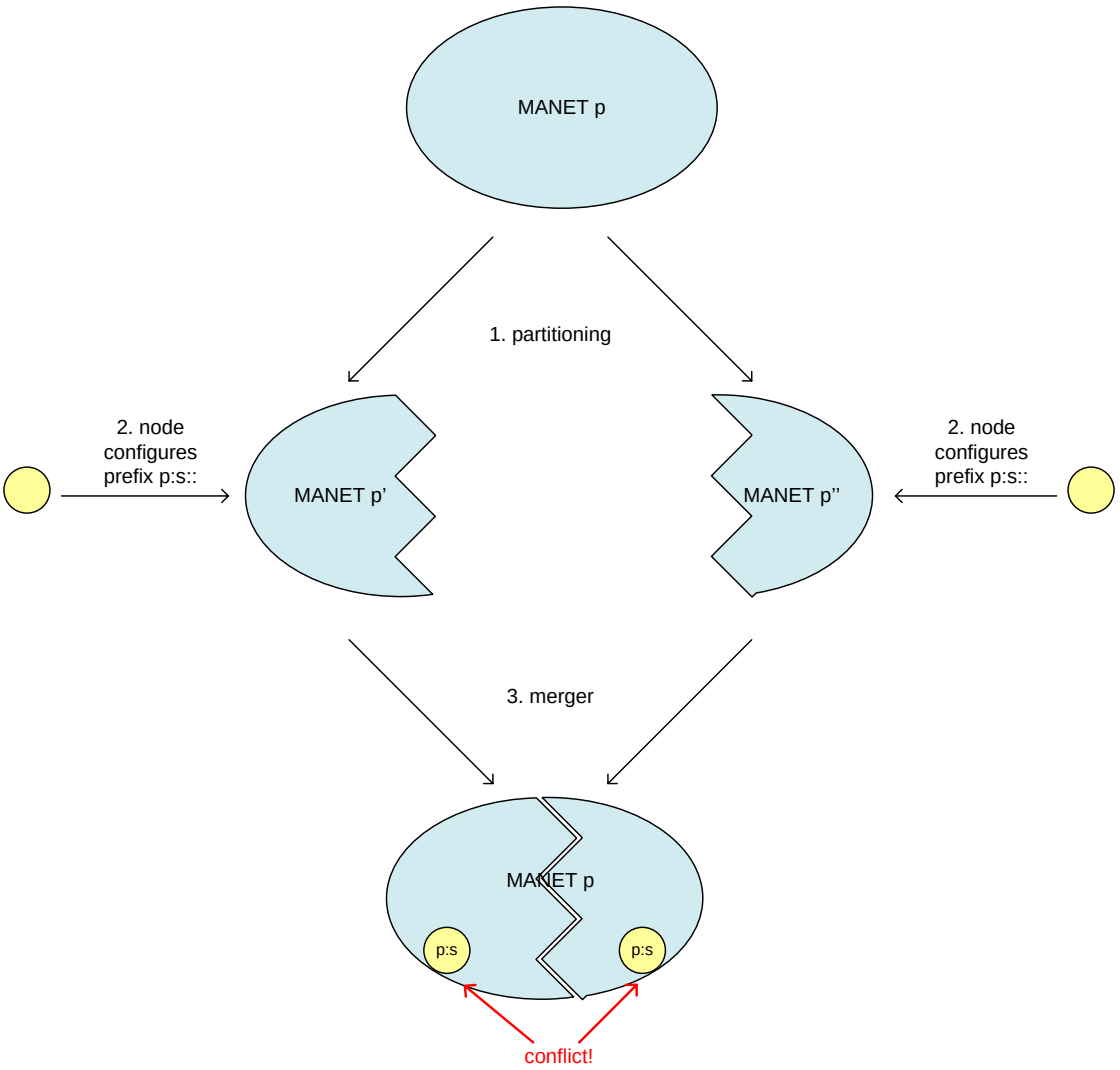


Figure 5.10: Autoconfiguration conflict in case of partitioning and re-merging

thesis, only some arising problems are sketched without trying to deliver solutions to these problems.

5.7.1 Duplicate addresses

Consider the following case which is depicted in figure 5.10:

A configured MANET with the common prefix part `p` splits into two parts `p'` and `p''`. Then for each MANET, a new node gets configured, both with the same prefix `p:s::/64`. So far this is legitimate, as there is no possible route from any node of `p'` to any node of `p''`. But as soon as these two MANETs join again, a conflict exists between the two nodes with the prefix `p:s::/64`.

Several possible solution to this problem were proposed (refer to section 4.2.2). Most of them are called **passive** duplicate address detection and try to passively detect collisions.

5.7.2 Prefix aggregation problem

Another problem occurs when creating routes using prefix aggregation (CIDR [42]) – refer to figure 5.11 for this example. After the MANET `p` has split into two parts `p'` and

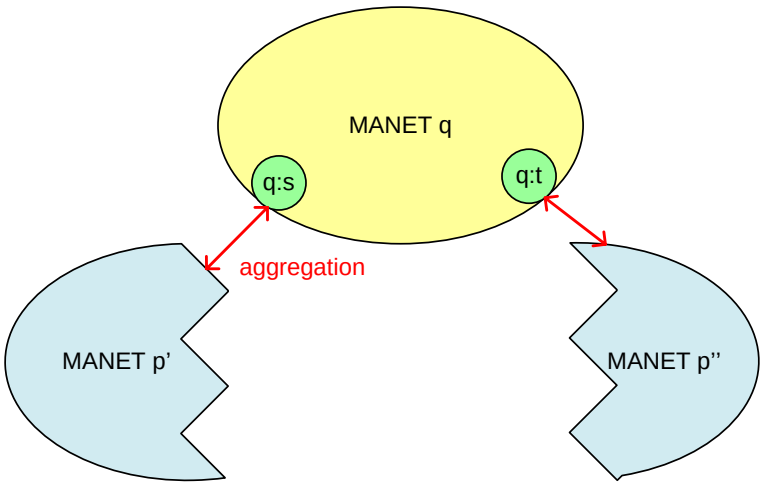


Figure 5.11: Problem when aggregating two split parts

p″, a third MANET q approaches. The two nodes q:s:: and q:t:: each create a route to the networks p′ and p″. However, as these two MANETs still have the same prefix p, the next-hop in the routing tables of nodes from q would be ambiguous.

5.8 Discussion & conclusion

The protocol proposed in this chapter is able to independently configure unique local and MANET-local prefixes based on the architectural model presented in section 2.5.1. This means that in particular *routers* are configured with unique prefixes and then attached hosts allocate IP addresses. The router prefixes are aggregatable with CIDR [42]. Moreover, the router configuration protocol is not based on link-local addresses of the nodes neither on a routing protocol – only link-local broadcasting is used. Deficiencies in terms of network overhead are addressed in chapter 6 with extensions and optimizations.

6 Extensions and optimizations of the router configuration

While the protocol proposed in chapter 5 is able to attribute unique prefixes and has very little preconditions (no routing protocol and no link-local addresses), it is not very efficient. In this chapter, several optimizations and extensions are proposed to increase efficiency, thus lowering network overhead. In very dense networks this is crucial for the autoconfiguration algorithm to work: if RS requests or RA replies are dropped due to collisions, duplicate prefixes can occur.

6.1 Performance optimizations

6.1.1 Jittering

Consider the case when at least two nodes are trying to be configured at exactly the same time. As they send the PS and RS messages always synchronously, all packets will be lost. To avoid this problem, jittering (as proposed in [19]) is introduced. Instead of instantly sending the PS and RS messages respectively, each node waits for a random duration of time. This time has to be considerably smaller than the `RS_TIMEOUT` and `PS_TIMEOUT` and should vary with every packet sent. Thus the probability of collisions is reduced on a large scale. Note that some routing protocols (like OLSR [22]) already implement jittering.

6.1.2 Proxying

Instead of broadcasting the RS messages all over the MANET, nodes can reply on behalf of a conflicting node if they already know that there will be a conflict (e.g. by having a route to the conflicting node in their routing table or by saving RS requests and RA answers for prefixes). A node can store prefixes for which it will reply with a RA message in a table. So whenever a RS request arrives at this node, it not only compares the requested prefix with its own prefix and the `configurating_nodes` table (refer to section 6.6.3) but also with all entries in this proxy table. An entry in this table consists of the prefix and a hash value of the RS message. This is necessary to avoid a conflict situation when the same node resends the RS request. However, for this to work, the RS messages of one node have to always use the same sequence number in order to produce the same hash value. The entries in this table should have an expiration time so that prefixes are not blocked forever. As soon as one prefix has reached the expiration time, it should be deleted from the proxy table.

Using proxies can save network congestion on a large scale. In particular when using link-state routing protocols like OLSR, proxying is easily doable as every node has a complete topological view of the MANET and thus knows all nodes (and their prefixes) in the MANET. For quantitative measurements of the savings refer to the simulation results in chapter 9.

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

6.1.3 Unicast RA messages

In the protocol presented in section 5.4, the RA message from a conflicting node is flooded throughout the MANET in order to be independent of any routing protocol. This, however, is not very efficient. A better way (also in combination with proxying as explained in section 6.1.2) is to use unicast for sending the RA message. In order to being able to send the RA message back to the initiator node that has sent the RS request, a temporary route back to the initiator has to be stored in the nodes' routing tables. Whenever a node receives a RS request, it stores the originating node as routing destination and the last node forwarding the RS message as next-hop in the routing table (similar to [68]).

A performance evaluation of using unicast together with proxying is presented in section 9.5.

6.1.4 Optimized broadcasting

As the RS message is not directly broadcasted throughout the MANET by the configurating itself but by the initiator node, broadcasting optimizations can be used. For instance, OLSR [24] as a routing protocol permits to apply optimized broadcasting using MPR selectors. This would not be possible if the configurating node itself had broadcasted the RS message as nodes using MPR broadcasting already need to have a valid IP address.

6.2 NHDP RS messages instead of MANET generalized format

Instead of using the RS messages as defined in section A.1, one might replace the RS and RA messages sent from the initiator node throughout the MANET by the standard RS and RA messages as defined in [62]. As no additional information like UUIDs have to be stored, this can be easily changed. It may make sense to unify packets sent throughout the MANET and using standardized packet formats instead of the message formats defined in this thesis. The reason for using NHDP messages is that they are already standardized and widely used within IPv6. Thus it is not necessary to reconfigure routers or firewalls to detect a new packet type. For the link-local communication between initiator and configurating nodes, it is however necessary to use the messages defined in section A.1 because NHDP messages do not offer enough extensibility (e.g. for specifying PS and PA messages and for including UUIDs).

6.3 Periodical prefix propagation

As an extension to the initial autoconfiguration algorithm it is proposed to have all configured nodes periodically broadcast PA messages in their 1-hop neighborhood every `PA_TIME_INTERVAL` ms. Whenever two MANETs meet, they can use the information gathered from the broadcasted PA messages and join the other MANET (refer to section 5.7 for merging and partitioning of MANETs).

If OLSRv2 [22] is used as routing protocol, the PA messages can be included in the TLV fields of HELLO messages. This limits network overhead.

6.4 IPv4 vs. IPv6

Another design goal in section 5.1 of the proposed algorithm was to be applicable in IPv4 as well as IPv6. The proposed algorithm was defined for IPv6 as it allows much larger MANETs when splitting an address into prefix parts and host parts. Another problem when using IPv4 is that the addresses contain semantics (like broadcast addresses). It is thus more complicated to design the algorithm and will not be considered in this thesis. However, when using UUIDs as proposed in section 5.4.1, IPv4 may be used as well for the algorithm without much change.

6.5 Initiator selection

In the autoconfiguration algorithm of section 5.4 step 3, the configurating node has to select its initiator if several PA messages arrive from different MANET nodes. It was proposed to select simply the first PA message that arrived. However, this may not be the best choice. The initiator nodes 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. This information could be included in free TLV fields of the PA message.

Examples of such information may be:

- number of nodes in the MANET
- density of the nodes in the MANET
- distance to an Internet gateway
- distance to a certain address prefix

6.6 Special issues

This section does not describe extensions with the objective to increase efficiency of the autoconfiguration protocol, but addresses several special issues that may occur in the protocol and proposes ways how to solve these problems.

6.6.1 Duplicate UUIDs

In this section, the case will be considered when two UUIDs are equal. This might seem quite improbable as the random numbers are 16 byte long. However, most devices equipped with a MANET interface will not have a cryptographical hardware pseudo-random generator which produces real random numbers. These devices rather implement a pseudo-random generator algorithm which bases on a seed and some environmental settings. When using several same devices with a cloned harddisk image, synchronized clocks and the same MAC address for instance, the different devices might create the same sequence of random numbers. If that is the case, duplicate UUIDs may appear.

In this section, this case for two configurating nodes and one initiator node will be discussed (as depicted in figure 6.1).

The only case when having duplicate UUIDs disturb the algorithm is when the initiator node sends an AC message to one of its configurating nodes and there are two adjacent configurating nodes that want to configure the same prefix. In the example both $config_1$ and $config_2$ send their RS messages for the same prefix – but not exactly in the same time as otherwise at least one of the packets would be lost. The initiator node will

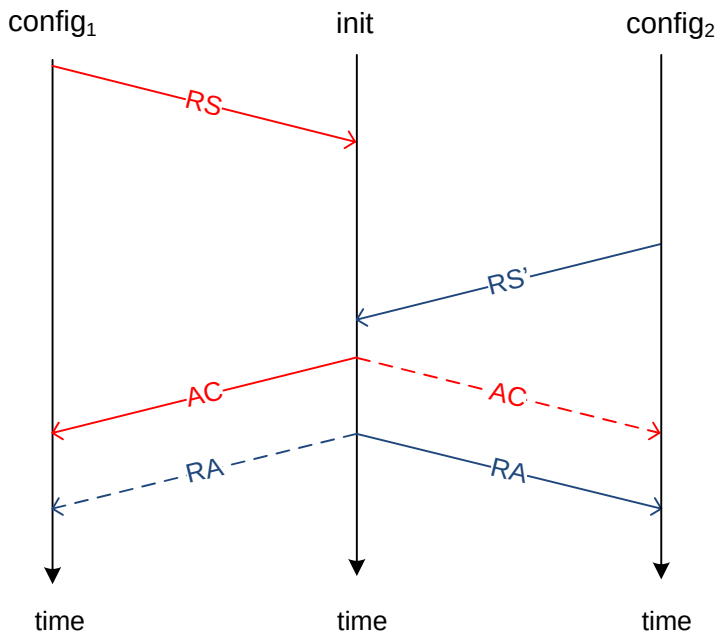


Figure 6.1: Two nodes $config_1$ and $config_2$ having the same UUID lead to ambiguous destinations of AC messages

send an AC message to $config_1$ (assuming there is no conflict in the MANET) and then the RA message to $config_2$. But as both configurating nodes have the same UUID, they will both receive the AC message and configure the duplicate address. The RA message arriving later will be ignored as both nodes have already been configured.

In the following, a solution for this problem is described which is depicted in figure 6.2. In this figure, the MAC layer and the application layer of each two configurating nodes are discerned. There are two possible reasons why two nodes choose the same UUID:

1. **The two nodes have differently seeded random number generators and simply create the same random number.**
   Even if this case is *extremely* improbable (refer to section 5.3.3), it is still considered here. Both nodes will probably send the first RS request after a different jitter delay t_0 and t_0' respectively. This time is now included in a TLV field of the RS message and stored on the initiator node. When the initiator node either sends an AC or RA message, it will again include the timestamp t_0 from the original RS message of that node. Both configurating node receiving the message will compare the included timestamp with their original timestamp t_0. If it differs, the autoconfiguration process will be aborted and the random number generator is seeded with the value $t_{recv} - t_0$ where t_{recv} is the arrival time of the RA or AC message. This will probably create different seeds on both nodes as the arriving time of the AC or RA message will be probably different at both nodes due to the different position and wireless environment. Now that both nodes have different UUIDs, the autoconfiguration process can be restarted normally.
2. **The two nodes 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 6.2, 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).

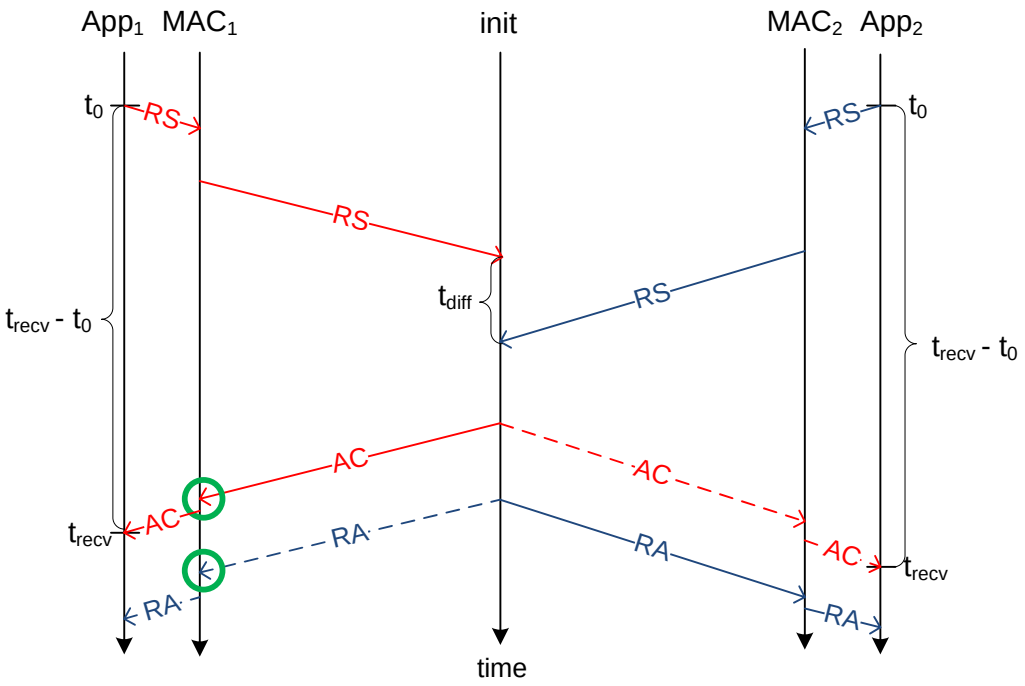


Figure 6.2: Two nodes $config_1$ and $config_2$ solving a UUID conflict

To avoid the UUID conflict, it does not help to include the timestamp t_0 in the RS message now as both nodes will use the same timestamp. But still, a UUID collision can be detected. For that to work, nodes have to wait a certain amount of time after receiving an AC or RA message before finalizing the autoconfiguration process. If a node receives an AC and a RA message for the same destination UUID within that short time interval (depicted as green circles in figure 6.2), it knows there must be a UUID conflict. Again, the random number generator is seeded with the value $t_{recv} - t_0$.

6.6.2 No initiator node exists

Another problem was detected when validating the algorithm using the model checker UPPAAL (refer to chapter 7). It emerges when two or more nodes want to configure an address almost simultaneously when there is not yet an already configured MANET node. This applies also when using jittering (refer to section 6.1.1). In figure 6.3 an example of that case is depicted. Both nodes $config_1$ and $config_2$ start sending their PS messages almost simultaneously. As there is no configured node yet, no PA replies will arrive. After `PS_MAX_TRIES` tries, both nodes will become initiator nodes but not being part of the same MANET (i.e. not having the same prefix part `p` and not having verified uniqueness of their addresses).

There can be two different cases as soon as the configurating nodes from the example become initiators and choose the random prefix `p:s::/64`:

1. If the nodes choose different prefix parts `p`, they cannot be aggregated using CIDR [42]. They may join the same prefix region by merging (refer to section 5.7).
2. If both nodes 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 (refer to section 5.7).

To solve this problem, configurating nodes can listen for incoming PS messages from other nodes. The node with the lower UUID waits at least `AC_TIMEOUT` seconds before

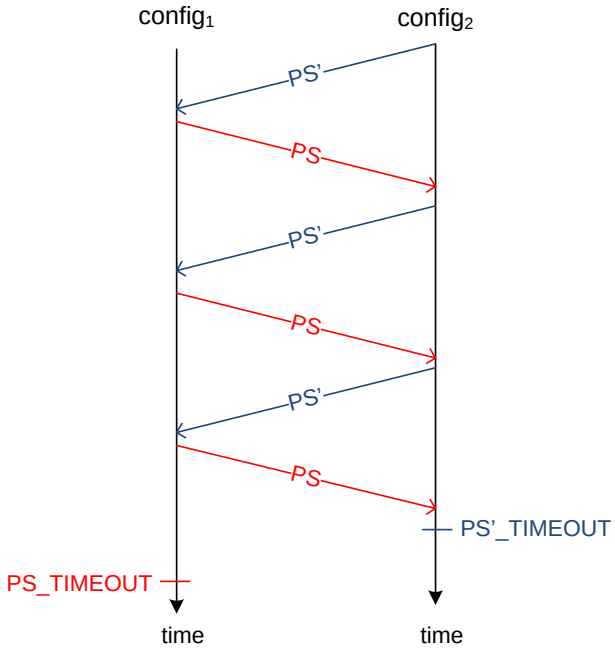


Figure 6.3: Problem when several nodes are configured almost simultaneously without initiator nodes

continuing the autoconfiguration process. Thus, the adjacent node can get configured in the meantime.

6.6.3 Initiator proxying

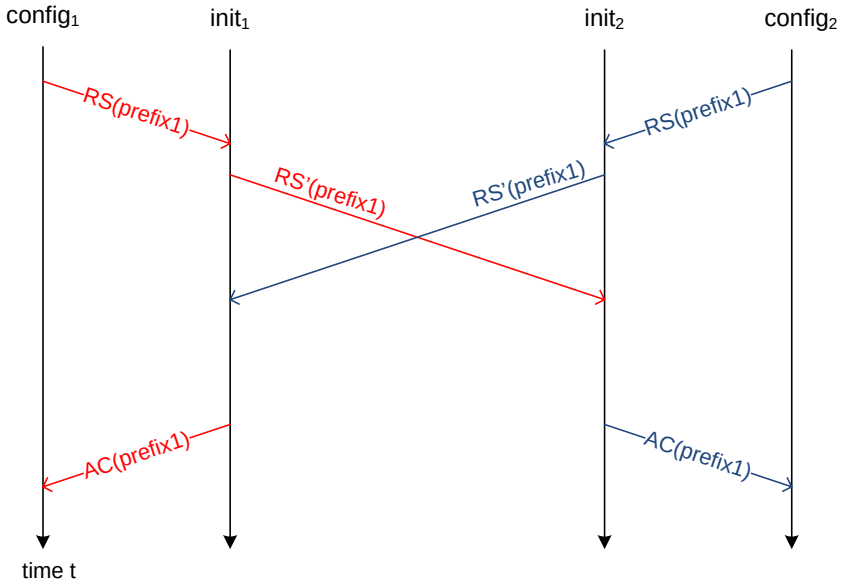
The following explanation is based on the example depicted in figure 6.4(a). Assuming four nodes of which two are already configured ($init_1$ and $init_2$) and two other nodes want to configure addresses ($config_1$ and $config_2$). $config_1$ and $init_1$ are adjacent and $config_2$ and $init_2$ as 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 nodes $config_1$ 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 node. But as in original algorithm of section 5.4 the tentative prefix from the RS message is only compared with the prefix of a node (and not with the configurating nodes of this initiator node), none of the nodes $init_1$ and $init_2$ will reply with a RA message. After the timeout, both initiator nodes will thus send an AC message confirming the tentative address to their configurating nodes.

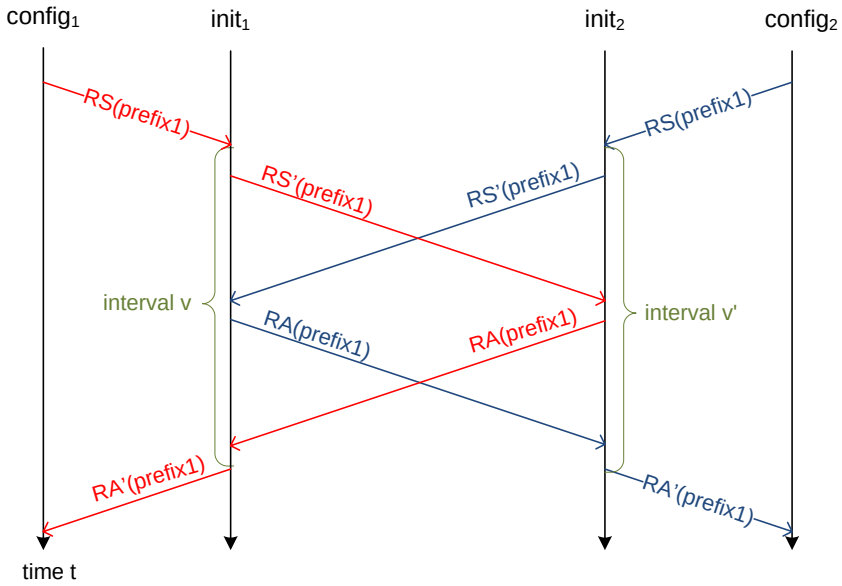
To avoid this allocation of duplicate addresses, the following is proposed (as depicted in figure 6.4(b)): As soon as the initiator node receives the RS request from the configurating node (and has no conflict with the initiator node itself), an entry in a table `configurating_nodes` is added which includes the tentative prefix of the configurating node and its UUID. For this to work the configurating node has to include its own UUID in the RS message. From the moment of the arrival of the RS message from the configurating node at the initiator node, the initiator node 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 message to the sending of either an AC or a RA message to the configurating node. In the example both initiator nodes will send RA messages back through the MANET when they receive the RS message through the MANET as they both have a table entry with the tentative prefix for their configurating node.

Note that the same algorithm applies also for the situation of only one initiator node which configures two adjacent configurating nodes almost simultaneously.



(a) Without proxying, two configurating nodes $config_1$ and $config_2$ could take the same address



(b) Proxying in the interval v avoids allocation of duplicate addresses

Figure 6.4: Initiator nodes serving as proxies for their configurating nodes

6.7 Conclusion

Several extensions have been proposed that can optimize the autoconfiguration protocol proposed in this thesis. Using jittering, proxying, optimized broadcasting and unicast messages can reduce the number of packets sent and the drop rate on a large scale. Some of these have been implemented in the simulation and the performance gain has been quantified. As a prospect, a detailed quantitative analysis including all optimizations could be performed.

In addition to the performance optimizations, several special issues that may arise during the autoconfiguration protocol have been addressed. It has been shown that the protocol even works when several nodes possess the same UUID.

7 Formal validation using the model checker UPPAAL

In the following chapter, a formal verification of the proposed algorithm of chapter 5 is presented. The motivation for using a model checker software for a formal validation was to prove (or disprove) the correctness of the protocol. Additionally, a model checker can explore cases where the protocol does not correctly work to find improvements.

In order to verify certain conditions of the algorithm, timed automatons are used in a model checking software. For the validation the freely available model checker UPPAAL is used as for its user-friendly GUI and as it has been successfully applied for other communication protocols (e.g. Zeroconf [36]).

According to the official web page [78], UPPAAL is an:

> "integrated tool environment for modeling, simulation and verification of real-time systems, developed jointly by Basic Research in Computer Science at Aalborg University in Denmark and the Department of Information Technology at Uppsala University in Sweden. It is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables (...). Typical application areas include real-time controllers and communication protocols in particular, those where timing aspects are critical."

In the following sections the proposed algorithm from chapter 5 is modeled in UPPAAL. Then some formal requirements and scenarios are validated in UPPAAL.

7.1 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**
  The initiator node will instantly broadcast a RS request to all other configured nodes as if they all were in a one-hop range. Thus, no node in the MANET has to forward an RS message.
- **No two nodes are configured in the same time**
  If two nodes configure in the same time, and there is no initiator node, no RA or AC message exchange can be performed and both nodes would become initiator nodes without having 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 6.6.2).
- **Nodes do not have a MANET prefix**
  For simplification reasons, nodes only have addresses of the form x where $x \in \mathbb{N}$. As all nodes are assumed to be in the same MANET, it is not necessary to care about the MANET prefix. The address is modeled in UPPAAL using a type definition called UUIDType which is defined as `scalar[k]` where k is the number of hosts.

7.2 Router autoconfiguration algorithm in UPPAAL

UPPAAL is based on the theory of timed automatons [2] and offers additional features like bounded integer variables and urgency [8]. The query language of UPPAAL, used to specify properties to be checked, is a subset of CTL (computation tree logic)[1].

The autoconfiguration algorithm presented in chapter 5 is partly based on [36]. It consists of three automatons: `Config`, `Network` and `AC_Timer`. In the following, these three automatons are explained in detail.

7.2.1 Config automaton

The `Config` automaton represents a mobile node (as depicted in figure 7.1). At the initial state $init$ the node is unconfigured until it reaches the state $isinitiator$. All states on the left-hand side of $isinitiator$ may occur during the configuration while all states right to that state may be entered when this node is already configured.

A node starts at the state $init$ and may only proceed to the next state $wait$ if no other node is currently configured (using a boolean variable $blocked$). It then sends several PS messages synchronizing with the `Network` automaton using the channel $send\_broadcast$. If no answer has been received after `PS_MAX_TRIES` tries, the automaton continues in the state $finished$, selecting a new IP address non-deterministically and reaching the state $isinitiator$. If, however, the node receives a PA message from the state ps_send, it chooses a new random address and tries to send it to the network in the state rs_send. From this state, there are three possibilities to proceed:

1. **A RA message has been received:**
   The automaton continues in the state ra_recvd, stops any AC timer and goes back to pa_recvd.
2. **An AC message has been received:**
   The automaton chooses the tentative IP address as permanent and continues at $isinitiator$.
3. **No message has been received after `RS_MAX_TRIES` tries:**
   The automaton restarts at the state $init$ aborting any AC timer running.

After a node has been successfully configured, it listens for incoming messages. As the node should react instantly on any incoming message, all following states are committed which means that time does not pass when moving from one state to the next. Several incoming message types can be discerned:

- **PS message:**
  The automaton sends a PA message including its own UUID synchronizing with the `Network` automaton by the $answer$ channel (transition ps_recvd_init –> $isinitiator$).
- **RS message:**
  Only RS messages are considered that are broadcasted throughout the MANET (i.e. the sender IP address is different to zero) or link-local RS messages (i.e. sender IP is zero and target UUID is the UUID of the current configurating node). The automaton checks whether the tentative IP address from the RS message conflicts with its own address (state rs_recvd_init). The automaton reaches either state $no\_conflict$ or $conflict$.
  1. **no conflict**
     If the node is not the configurating node but any node in the MANET receiving a broadcasted RS message, it returns to the $isinitiator$ state without

1 In contrast to CTL it does not offer nesting of path formulae

any action. If the node, however, is the initiator node for the node sending the RS message, it first broadcasts the RS message in the MANET (using the $answer$ channel, transition $timer\_started$ –> $start\_manet\_broadcast$), and then starts the AC timer by synchronizing with the `AC_timers` automaton (transition $start\_manet\_broadcast$ –> $isinitiator$).

2. **conflict**
   If the node is not the configurating node but any node in the MANET receiving a broadcasted RS message, the automaton continues in the state $manet\_conflict$, broadcasts a RA message throughout the MANET, and then reaches the state $isinitiator$. If the node, however, is the initiator node for the node sending the RS message, it reaches state $local\_conflict$, broadcasts a RA message to the configurating node, and finally goes to state $isinitiator$.

- **RA message:** If a RA message comes from the MANET and arrives at the initiator node, the automaton continues at the state $local\_conflict$ (see above for actions performed from this state).

7.2.2 Network automaton

The `Network` automaton as depicted in figure 7.2 simulates the behavior of the sending of messages and replies to these messages. All messages and replies are sent as broadcast messages.

Starting from the state $idle$, the automaton waits for an incoming broadcast request (via the channel $send\_broadcast$). It then chooses randomly a target node as destination for the packet and synchronizes with that host using the urgent broadcast channel $receive\_msg[UUIDType]$. Any host may reply directly using the transition $sent$ –>* $reply2$ or from the transition $ll\_broadcast$ –>* $reply1$. It is assured that all nodes receive the broadcast message and may answer using the two functions $all\_sent()$ and $all\_replied()$ respectively.

7.2.3 AC_Timer automaton

The `AC_Timer` automaton (depicted in figure 7.3) is responsible for triggering the timer for the AC message that has to be sent from an initiator node. The timer is activated using the $ac\_timer[UUIDType]$ channel. Then, the automaton waits for ac_delay seconds. If within this time the timer is not interrupted using the channel $abort\_ac[UUIDType]$, the AC message is broadcasted using the $send\_broadcast$ channel (refer to section 7.2.2).

7.3 Verification

Several statements have successfully be proven correct:

- **no deadlock**
  This assures that never should a deadlock occur. A deadlock is a state where there is no action successor.
- **no two nodes ever have the same IP**
  This is an obvious goal of autoconfiguration. A node should either not yet be configured or if it already has been configured, no other node should have the same address.
- **there exists a path to a state where all nodes are configured**
  This proves that it is possible to configure all nodes correctly.

The last statement, however, is weaker than proving that all nodes *must* be configured after a finite time. But for this to work, CTL path quantifiers must be nested which is not possible in UPPAAL. Due to the limited time of the thesis, it has not been explored 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.

7.4 Conclusion

Testing network protocols with a model checker validated formally some of the requirements of the protocol and showed cases where an initial version of the protocol did not work (refer to section 7.1). It has been shown that the protocol never wrongly configures addresses. However, it could not be shown that the protocol will always attribute addresses to all nodes due to a limitation in the language of the model checker. As a perspective, reasons to the limitation and ways to prove the statement could be explored.

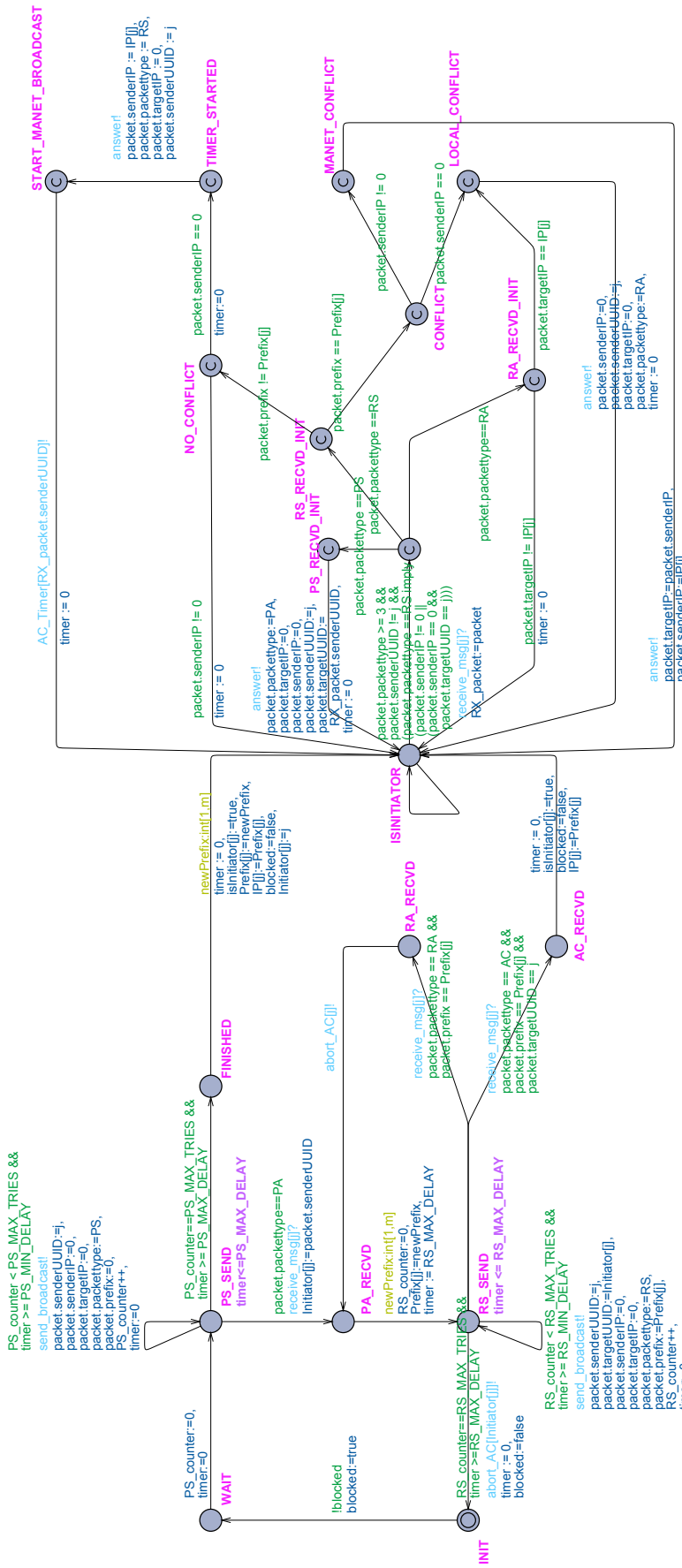


Figure 7.1: The automaton representing a mobile node

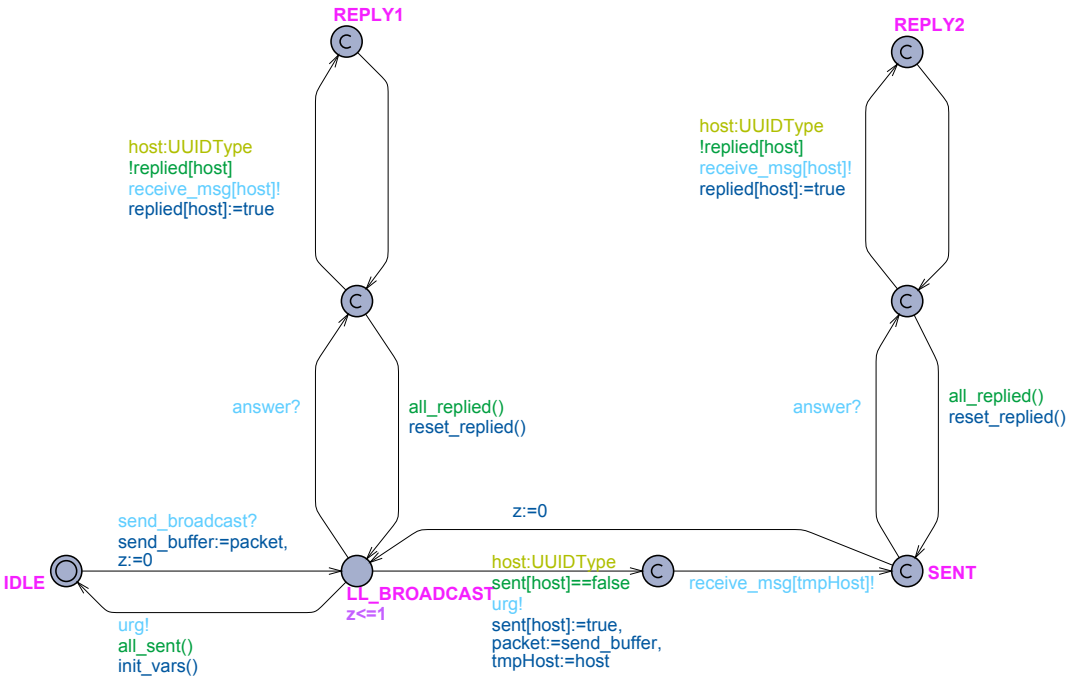


Figure 7.2: The automaton representing the network layer

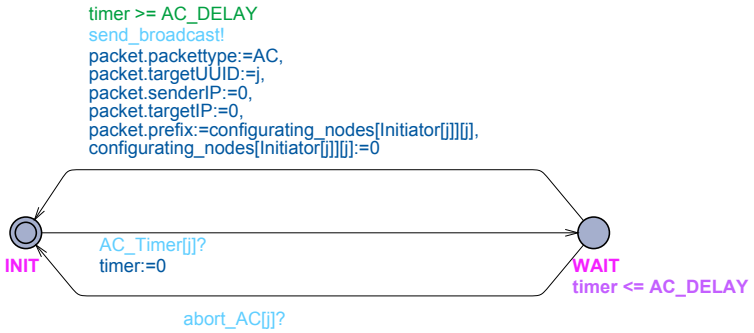


Figure 7.3: The automaton representing the timer for AC messages

8 Implementation / Testbed

After a formal validation (refer to chapter 7) of the autoconfiguration protocol specified in chapter 5, an implementation of the protocol is presented in this chapter. The implementation was verified with a testbed running on several Nokia N770 devices[1].

8.1 Introduction to Nokia 770

The Nokia N770 (depicted in figure 8.1) is a PDA running a complete Debian Linux distribution on an ARMEL processor. Due to its small size, WiFi interface and the Linux platform, it is very suitable for testing any MANET protocol. Hipercom has 10 of these N770 for testing purposes. In the following subsections, preconditions for the testbed of the autoconfiguration protocol are presented.

Figure 8.1: The Nokia N770

8.1.1 Operating system and kernel patches

In order to run the implementation (presented in section 8.2), the operating system and the kernel had to be patched using the `flasher` utility offered at [28]. The operating system was patched to the OS 2006. The kernel had to be patched in order to enable IPv6 and ip6tables (refer to 8.1.3). As the N770 is operating on an ARMEL processor, a cross-compilation toolkit called Scratchbox was used to compile the new kernel (as described in section 8.1.2).

8.1.2 Scratchbox cross-compiler

For compilation of software for the N770, the cross-compiler "scratchbox" [28] was used in version 2.2 ("gregale"). It offers a complete cross-compiler for Linux based on

1 http://europe.nokia.com/770

GCC [37]. Using scratchbox avoids compiling software on the Nokia N770 which is time-consuming as the processor speed is quite limited. Note, however, that often precompiled software is available from [28] and may often be installed using the `apt-get` tool which resolves all dependencies as well.

8.1.3 Iptables

For creating easily a multi-hop wireless network, iptables (or its IPv6 equivalent "ip6tables") with MAC address source filtering is very useful if one wants to avoid spreading the Nokia N770s over the whole campus. With ip6tables it is possible to create a rule to drop packets from a certain source MAC address. For this to work on the Nokia N770, however, ip6tables must be recompiled from the source and the kernel has to be patched as well (refer to section 8.1.1).

8.1.4 OLSR daemon

As described in section 5.4, RA messages can be sent back via unicast to the originating initiator node instead of flooding them throughout the MANET. For this purpose, the OLSR.org client was installed. It permits an easy usage and offers the possibility of writing plugins. For example, a plugin could be written that broadcasts the RS messages using MPR relaying.

8.1.5 Further prerequisites

For convenience, a SSH server, IPv6 capable shell and automatic start of the wireless interface in the ad-hoc mode can be installed.

8.2 Autoconfiguration agent implementation

The implementation is based on a prototype library called "Protean Protocol Prototyping Library (ProtoLib)" [52]. It offers classes for IP addresses, sockets, packets, timers, listeners, event notifiers and so on, supporting a variety of operating systems and network simulators (NS2 and Opnet). In particular, it includes classes for the MANET generalized packet/message format [21]. With ProtoLib, a prototype of the autoconfiguration protocol proposed in chapter 5 was implemented which could be adapted without many changes for the NS2 simulator (refer to chapter 9).

The protocol was designed as close as possible to the specification of section 5.4 (depicted in figure 5.5 and 5.6) and will thus not be described in detail here. However, the usage of the autoconfiguration client will be documented in the following.

8.2.1 Command line parameters

The autoconfiguration client can be called from a Linux shell with some optional parameters:

```
autoconf
   [--first-prefix-part d | -F d]
   [--prefix s | -P s]
   [--interface iface | -I iface]
```

- The `first-prefix-part` parameter specifies the first part of a node prefix (i.e. the site prefix `d` within a prefix `d:s:p::`). It is four characters long and must be written as a "hexadecimal" string. When no argument is given, the default value `fec0` is used.
- The `prefix` parameter specifies the MANET prefix. Normally, this prefix part is created randomly when the first node in a MANET becomes initiator. It is `NODE_PREFIX_LENGTH` characters long (as specified in autoconf.h).
- The `interface` parameter specifies the MANET interface. The default value is `wlan0` and is defined in `IFACE_NAME` in autoconf.h.

The autoconfiguration client has to be executed as root and expects two programs to be in the PATH: olsrd (the OLSR.org daemon) and ifconfig (for adding the IPv6 MANET address).

8.2.2 Testbed

In this section, the environment of the test setting and the runs themselves are described. Starting with an experiment with only one node, five nodes were used in the final experiment. For creating a multi-hop network, a MAC address blocking script has been written which had the following content:

```
ip6tables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:XX -j DROP
```

The autoconfiguration agent was then started consecutively on the nodes. In all following cases, prefix collisions have been successfully detected and new unique prefixes were chosen.

Run with only one node

The first test was to run the autoconfiguration agent with only one node (as depicted in figure 8.2).

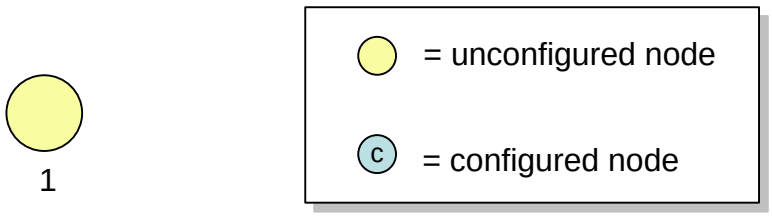


Figure 8.2: Autoconfiguration test with one node

The following output of the autoconfiguration agent is displayed on the screen:

```
herberg@node1:~/protolib/unix> sudo ./autoconf
Interface wlan0 detected
Local address: fe80::215:f2ff:fe05:896d
Listening on UDP port 54322
* sending PS message to ff02::1
* sending PS message to ff02::1
* sending PS message to ff02::1

-------- Node has completed autoconfiguration -------
* choosing new random prefix and listening on port 54321
```

```
* new IP Prefix: fec0:917a:53aa:4a43::
* launching OLSR daemon with pid 25472
```

The first node starts the autoconfiguration client on its link-local interface `fe80::215:f2ff:fe05:896d`. It sends three PS messages to the broadcast address `ff02::1` without receiving an answer. It then chooses and adds a site-local prefix `fec0:917a:53aa:4a43::` to the interface and launches the OLSR daemon. This new IP prefix consists of the pre-defined site prefix part `fec0`, a random MANET prefix part `917a`, and a random node prefix part `53aa:4a43`.

Run with two nodes

In this case (depicted in figure 8.3), a second node enters the MANET created above. This output is created on the new node:

```
herberg@node2:~/protolib/unix> sudo ./autoconf
Interface wlan0 detected
Local address: fe80::204:76ff:fe13:e751
Listening on UDP port 54322
* sending PS message to ff02::1
* PA message received from fe80::215:f2ff:fe05:896d
Initiator: fe80::215:f2ff:fe05:896d
Prefix: fec0:917a
Tentative prefix: fec0:917a:2d6e:9834::
* sending RS message to ff02::1
* AC message received from fe80::215:f2ff:fe05:896d
* Setting the prefix fec0:917a:2d6e:9834:: as permanent.

-------- Node has completed autoconfiguration -------
* new IP Prefix: fec0:917a:2d6e:9834::
* launching OLSR daemon with pid 32431
```

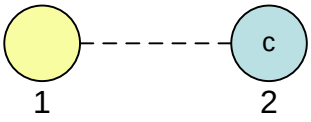


Figure 8.3: Autoconfiguration test with two nodes

After receiving a PA answer from the initiator node including the MANET prefix `fec0:917a`, the configurating node sends the RS message to the link-local broadcast address. As no conflict occurs, the initiator node sends the AC message after the time-out, and the configurating node sets the new prefix as permanent, launching the OLSR daemon.

Run with three nodes

In figure 8.4, a test with three nodes in a line can be seen. Node 2 and 3 are already configured. Now, node 1 chooses the same prefix as node 3 (via the command line option `-prefix`) which is a 2-hop neighbor of node 1.

The RS message arriving at the initiator node is broadcasted through the MANET and reaches node 3 which detects a conflict. It sends back the RA message to the initiator node which forwards it to its configurating nodes in a 1-hop range. The configurating node chooses a random new node prefix and includes it in a new RS request.

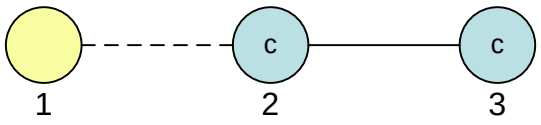


Figure 8.4: Autoconfiguration test with three nodes

Run with four nodes

This case (depicted in figure 8.5) is different to the case above in one respect: node 3 has to forward the RS request to node 4 for the broadcast. The implementation includes a table storing hash values of recently seen messages (as implemented also in OLSR [24]). This is necessary to avoid broadcast storms. This implementation part could be avoided if the optimization mentioned in section 6.1.4 would be used. A plugin could be written for the OLSR daemon to use MPR flooding. Alternately, the existing BMF plugin [77] could be modified to run on IPv6. However, it encapsulates the messages within an OLSR header so that the message format proposed in section A.1 would be hidden. This may be disadvantageous in case when NHDP messages [62] are preferred for the RS / RA messages in the MANET instead of using the generalized MANET message/packet format [21]. Note that in the implementation the RA message from node 4 is sent via

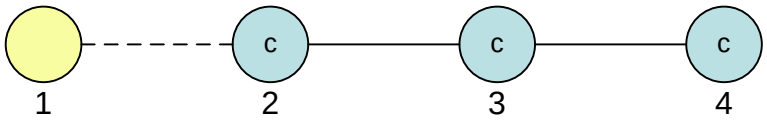


Figure 8.5: Autoconfiguration test with four nodes

unicast to the initiator node.

Run with five nodes in a row

With five nodes in a row (depicted in figure 8.6) is basically similar to the run with four nodes in a row described above.

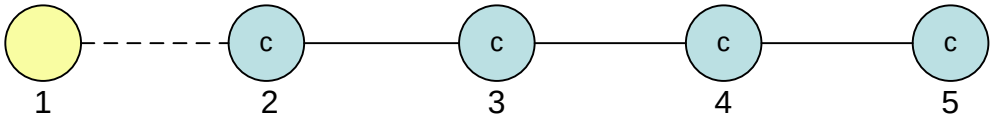


Figure 8.6: Autoconfiguration test with five nodes in a row

Run with five nodes in a diamond

In this case (see figure 8.7), again five nodes were used. However, nodes 3 and 4 are both 1-hop neighbors of nodes 2 and 5. This has no influence on the correct duplicate address detection.

8.3 Conclusion

The implementation of the autoconfiguration protocol and the successful tests of the implementation have shown that the protocol is working in real-life situations. In the testbed, up to five nodes could be correctly allocated with unique prefixes.

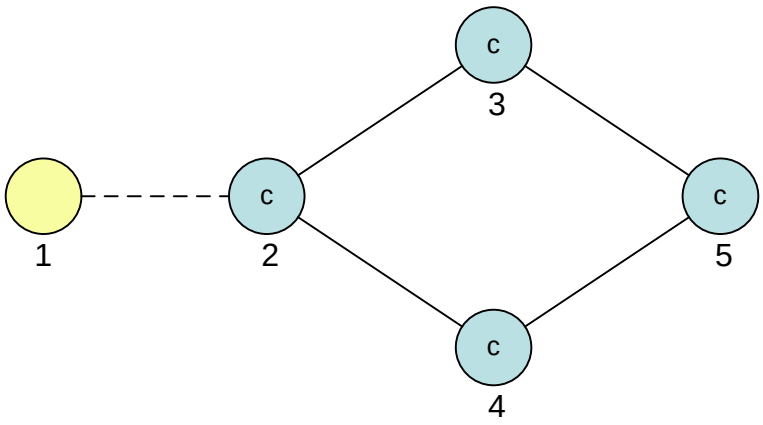


Figure 8.7: Autoconfiguration test with five nodes in a diamond

As a perspective, more nodes could be used in a testbed. For this testbed to be set up quickly, however, an automated way of using the ip6tables MAC address blocking script of section 8.2.2 should be accomplished. Otherwise, it takes a long time blocking all desired MAC addresses on each device for creating the multi-hop network.

9 Simulation

In chapter 8 it has been shown that the implementation of the autoconfiguration protocol specified in chapter 5 was capable of attributing unique prefixes to nodes in a MANET. However, it is time consuming and expensive to set up a testbed with a larger number of nodes. Thus, a common way of testing protocols in MANETs is using a network simulator. In this chapter, the implementation of the autoconfiguration protocol specified in this thesis in the network simulator NS2 [34] is presented after a general introduction to MANET simulation. Then the used settings and results of the simulations are presented and concluded.

9.1 Introduction to MANET simulation

Many papers in the area of MANETs use network simulations to verify their proposed protocols, and to test and to compare their performance to other protocols. According to a survey conducted by [50] on the MobiHoc symposium [35], over 75% of all papers published used simulation to test their research. Simulation enables researchers to test their protocols in a repeatable way with several dozens to hundreds of nodes.

However, it is not always obvious whether the simulations results correspond to the real behavior of networks. [12] has shown that different simulators often produce very different results when using the same protocol and scenario. They have tested three famous simulators (OpNet, NS2, GloMoSim) with a simple flooding protocol and noticed important divergences between the simulators and, in some cases, results that are barely comparable. Another problem is the stated advantage of repeatability of such experiments: As researchers often do not publish their source code, the only way of reproducing such an experiment is using the same scenario parameters is they present in their papers. According to [50], less then 15% of the papers published in [35] were repeatable due to missing declaration of parameters in their papers.

To avoid these repeatability problems in this thesis, all parameters mentioned in [50] for being able to replicate the results are presented and the source code will be available for download on [39].

9.2 The NS2 simulator

In this thesis, the NS2 simulator [34] was chosen for the simulation. The reason for this choice is that first of all NS2 is freely available for educational purposes. Moreover, most papers concerning MANETs use NS2 as the simulator, so simulation results are easier to compare. Refer to figure 9.1 for an overview of the simulators and a rough guess of their proportion in MANET papers.

According to [34], "Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks". Together with NAM (a network animator program included in NS), it allows simulation, tracing, but also visualization of networks.

NS2, but also new agents (for example routing protocols but also the autoconfiguration agent) are implemented in C++. However, scripts for movements, node configurations,

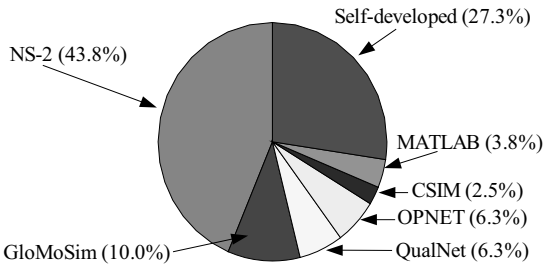


Figure 9.1: Simulator usage (source: [50])

message exchange and so on are specified in OTcl, a simple scripting language. Code from both languages can interact which makes implementation sometimes not very straight-forward.

9.3 Implementation details

In this section, modifications to the OLSR routing agent are described and finally the autoconfiguration agent is explained.

9.3.1 Routing protocol

As specified in section 5.4.2, the autoconfiguration protocol presented in this thesis is independent of the underlying routing protocol. In fact, for the simulation, no multi-hop routing protocol has been used. Only broadcasting in a 1-hop neighborhood is applied. [72] was used as routing protocol, but all HELLO and TC message exchanges were disabled, so the routing tables were always empty. The reason for not just taking the "DumbAgent" which is delivered with NS2 was to easily add static routes for an optimization which is explained in section 9.5. The changes to the OLSR agent can be found in section A.3 as a `diff` generated file.

9.3.2 Autoconfiguration agent description

As explained in section 8.2, a prototyping library called ProtoLib [52] was used as underlying framework. This framework enables the simulation of the code as an agent in NS2 and OpNET by changing some compile flags. However, some smaller changes had to be implemented in the agent itself in order to reflect the different addressing scheme of NS2, and the different seeding of random numbers.

Addressing nodes in NS2

While in the real implementation presented in chapter 8, real IPv6 addresses were used, NS2 uses a different addressing scheme. Basically, it uses 32-bit long integers as node identifiers. To reflect these different addresses, the C++ class representing an IP address in ProtoLib offers the possibility to use so-called SIM addresses instead of IPv6 or IPv4. These addresses are fixed at the creation time of a node and no addresses can be added as it was the case in the real implementation when an IPv6 site-local or global address was added when completing the autoconfiguration. So during the simulation, only the SIM addresses are used and the IPv6 addresses are only stored in a local variable of each node which will be printed out to a trace file at the end of each node's autoconfiguration process.

Random numbers in NS2

For creating random UUIDs and prefixes for each node, a correct seeding of random values is crucial and often not correctly described or performed in papers (refer to [50]). NS2 uses the so-called RNG class based on an implementation of the combined multiple recursive generator MRG32k3a proposed by [53]. The following description is taken from the NS2 manual [34]:

"The MRG32k3a generator provides $1.8 \cdot 10^{19}$ independent streams of random numbers, each of which consists of $2.3 \cdot 10^{15}$ substreams. (...) A default RNG (`defaultRNG`), created at simulator initialization time, is provided. If multiple random variables are used in a simulation, each random variable should use a separate RNG object. When a new RNG object is created, it is automatically seeded to the beginning of the next independent stream of random numbers. Used in this manner, the implementation allows for a maximum of $1.8 \cdot 10^{19}$ random variables."

By default, the RNG generator is seeded with 12345. Note that when seeding the RNG object with the value 0, the current time of the computer is used as seed thus providing independent streams for several runs – however, this makes repeatability impossible, and even worse, there is no guarantee that the streams produced by two random seeds will not overlap. For assuring independent runs in the implementation presented in this chapter, the random number generator is seeded with the `RNG::PREDEF_SEED_SOURCE` directive, and the number of the run. This assures non-overlapping streams on the one hand and repeatability on the other hand.

In the simulation of the autoconfiguration agent, a new RNG object is used for each variable and each node uses the x-th substream where x is the NS2 ID of the node.

9.4 Simulation settings

In this section, the simulation environment and settings are described. Refer to table 9.4 and table 9.5 for an overview of the settings used for the simulation. The selection of the parameters in the tables are based on [50].

9.4.1 Mobility scenario generation

The mobility scenarios are created with the mobility scenario creator from CMU called `setdest` which is included in NS2. The usage of the program is:

```
./setdest -n <num_of_nodes> -p <pausetime> -s <maxspeed>
          -t <simtime> -x <maxx> -y <maxy>
           > <outdir>/<scenario-file>
```

For the simulation of the autoconfiguration protocol proposed in this thesis, a field of 1000 m x 1000 m has been used, a simulation time of $number\_of\_nodes + 20$ seconds and no mobility. The reason for not including mobility is that conflicts deriving from partitioning and merging of MANETs would arise. As in this thesis no handling of merging and partitioning is proposed (refer to section 5.7), it does not make sense to simulate mobility. 20 different scenarios have been created for each 20, 40, 60, 80 and 100 nodes.

9.4.2 Wireless settings

Refer to table 9.1 for the values used for the wireless interface used by the NS2 command `node-config`. The antennas are omni-directional and positioned 1.5 meters

Parameter	Value
llType	LL
macType	Mac/802_11
ifqType	Queue/DropTail/PriQueue
ifqLen	50
antType	Antenna/OmniAntenna
propInstance	Propagation/TwoRayGround
phyType	Phy/WirelessPhy
channel	Channel/WirelessChannel
wiredRouting	OFF

Table 9.1: Wireless settings

Parameter	Value
X_	0
Y_	0
Z_	1.5
Gt_	1.0
Gr_	1.0

Table 9.2: Antenna settings

above each node (refer to table 9.2). The wireless interface is set up with parameters to make it work like the 914MHz Lucent WaveLAN DSSS radio interface (refer to table 9.3).

9.4.3 Automated testing

In order to launch easily several runs of the simulation, two Tcl scripts have been written to automate the test process (find the scripts in the appendix, section A.2.1 and A.2.2). The `autoconf.tcl` script first creates the mobility scenario, then launches NS2 x times in a row executing the second script `autoconf-wifi.tcl`. Afterwards, the output file is checked for duplicate prefixes. The script is launched with

```
./autoconf.tcl -nn <number_of_nodes> -run <number_of_runs>
```

To count the number of sent packets, one can use the `grep` command from Linux:

```
grep -c "^s[0-9a-zA-Z. -]*MAC[0-9a-zA-Z. -]*-I" \
  autoconf-<nn>-<run>.tr
```

This counts all IP packets sent from all nodes. Note that MANET broadcast packets are counted several times (for each node forwarding the packet).

9.4.4 Agent starting

The autoconfiguration agent it started consecutively on each node every second, e.g. the node with the ID 17 enables autoconfiguration at 17.0s after simulation start. Note that the nodes are randomly distributed on the scenario field. Each agent first acquires the MANET prefix either by getting a Prefix Advertisement (PA) message or by choosing a random one if it is the first node in the MANET. It then chooses the prefix `0:1:` after the MANET prefix. Thus, all nodes in the same MANET want to configure the same prefix `0:1:`. From the second node in a MANET, collisions will consequently appear.

Parameter	Value
CPThresh_	10.0
CSThresh_	1.559e-11
RXThresh_	3.652e-10
Rb_	2*1e6
Pt_	0.2818
freq_	914e+6
L_	1.0

Table 9.3: Further wireless settings

Parameter	Value
Simulator name	NS2
Simulator version	2.31
Operating system	Linux kernel 2.6.18.2 x86_64
Type of simulation	terminating
PRNG	MRG32k3a

Table 9.4: Simulator and environment

9.5 Simulation results

In the following, the results of the simulation are described. Two different versions of the autoconfiguration algorithm have been tested: the basic version presented in section 5.4 and an extended version with proxying (refer to section 6.1.2).

9.5.1 Collisions

In neither of the two versions did any collisions occur, but the number of exchanged packets and drop rates are different. The following exemplary output of the simulation corresponds to the scenario depicted in figure 9.2. The different MANETs can be discerned by the two bytes of the prefix after the common prefix `fec0`.

```
Node_(0):   fec0:e2d7:0:1::
Node_(7):   fec0:e2d7:3258:12ad::
Node_(11):  fec0:e2d7:8405:d4e8::
Node_(1):   fec0:e2d7:2597:174::
Node_(19):  fec0:e2d7:5bab:d87::
Node_(14):  fec0:e2d7:cb5b:a813::
Node_(16):  fec0:e2d7:62ec:1ea::
Node_(19):  fec0:e2d7:5bab:d87::
Node_(2):   fec0:3d51:0:1::
Node_(5):   fec0:3d51:1a8a:de16::
Node_(3):   fec0:3d51:c1a6:c2ec::
Node_(4):   fec0:3d51:e009:2497::
Node_(6):   fec0:3d51:b1eb:9816::
Node_(8):   fec0:d723:0:1::
Node_(13):  fec0:d723:29a8:ea84::
Node_(17):  fec0:d723:2dad:5b6d::
Node_(9):   fec0:7c25:0:1::
Node_(10):  fec0:ab37:0:1::
Node_(12):  fec0:ab37:973d:b53d::
```

Parameter	Value
Number of nodes	20 / 40 / 60 / 80
Size of simulation area	1000 m x 1000 m
Transmission range	250 m
Simulation duration	100 s
Traffic type	UDP
Number of simulation runs	20
Mean speed of the nodes	0 m/s
Speed variance about the mean	0 m/s
Mean pause time of the nodes	0 s
Pause time variance about the mean	0 s
Mobility model	Random waypoint mobility model

Table 9.5: Simulator input parameters

```
Node_(18):   fec0:ab37:15b5:4e91::
Node_(15):   fec0:ab37:d32d:90a4
```

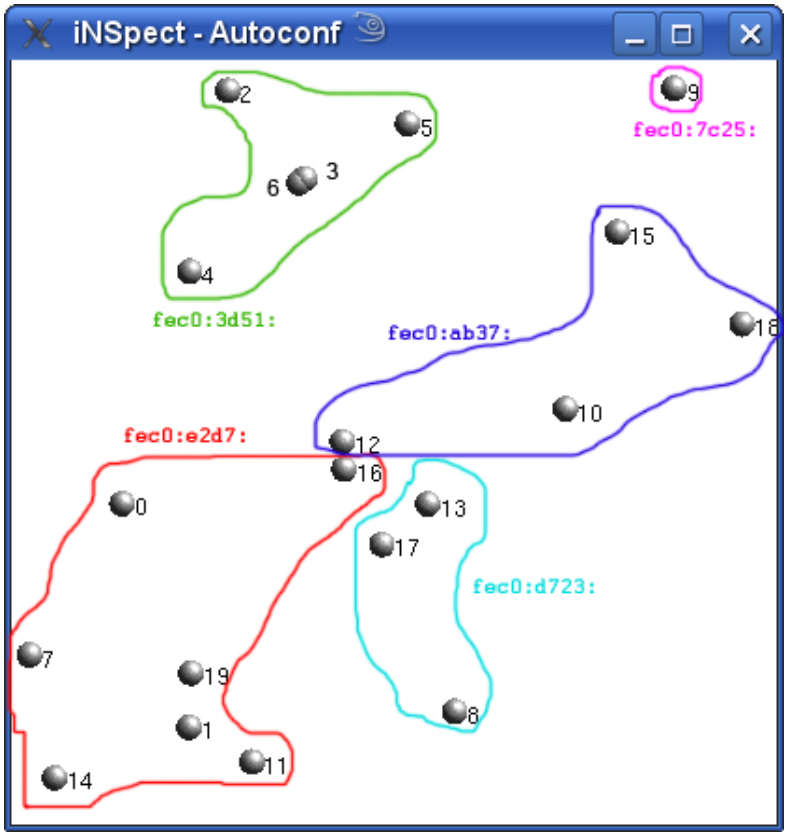


Figure 9.2: MANETs after autoconfiguration (displayed with [51] and edited afterwards)

9.5.2 Packet numbers and drop rates

In the basic version of the algorithm RS and RA messages are always simply flooded throughout the MANET (refer to section 5.4). However, in the proxy version, nodes store prefixes from RS broadcasted requests for the duration of the simulation[1]. In

1 In a real-life implementation, the proxy entries should expire after a certain time

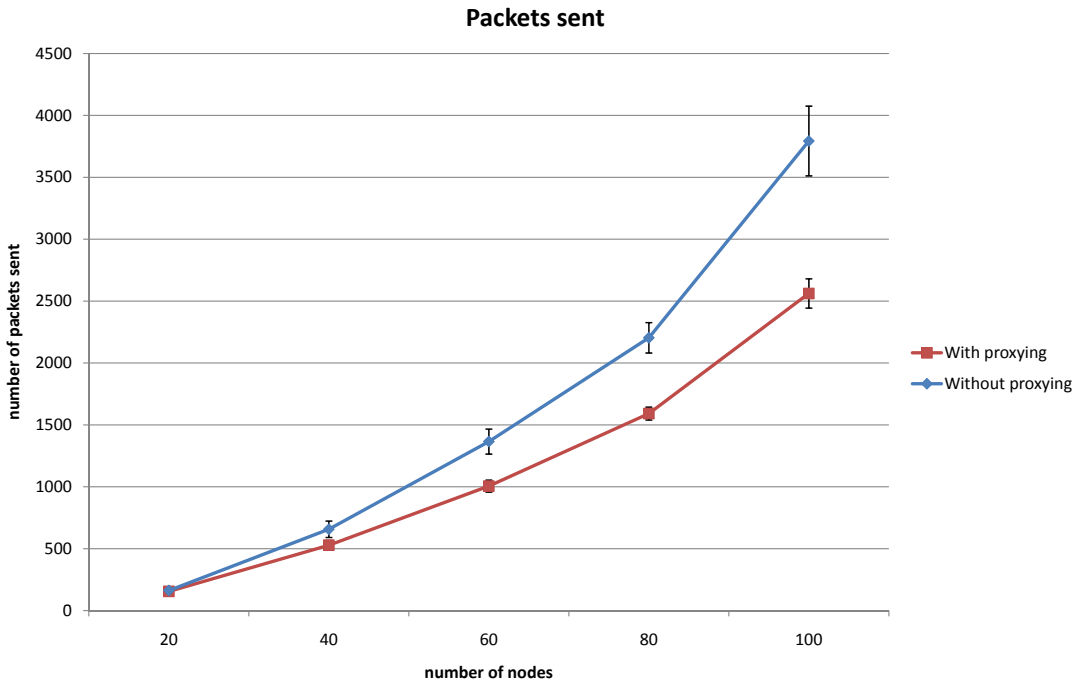


Figure 9.3: Number of sent IP packets of the autoconfiguration agent (90% confidence intervals)

addition, they store a route back to the originating node of the request. If they get another request for the same prefix and a different packet sequence number, they send a RA message back using unicast thus avoiding another broadcast.

For the following results, each test has been repeated 20 times taking the average values. Figure 9.3 depicts the number of sent packets in both versions. Note that this number does not include ARP or MAC layer packets (RTS, CTS, ACK, etc.) which have been sent in the proxy-version. As can be seen in the figure, when using proxying, significantly less packets are exchanged. Consequently, the drop rates depicted in figure 9.4 are lower as well.

9.6 Conclusion and prospect

The simulation has demonstrated that the protocol correctly configures prefixes to nodes, also with a bigger number of nodes. It has also been proven that the protocol does not depend on a multi-hop routing protocol – only link-local broadcast is necessary. Using a proxying mechanism as described in section 6.1.2 decreases the number of exchanged packets and also the drop rate. However, the drop rate is still quite high, so prefix collision may occur when the RS request or RA message are dropped. To decrease the number of packets, an optimized broadcast mechanism could be implemented as it is for example used in OLSR [24]. The routing agent must be extended to forward RS messages using MPR relaying.

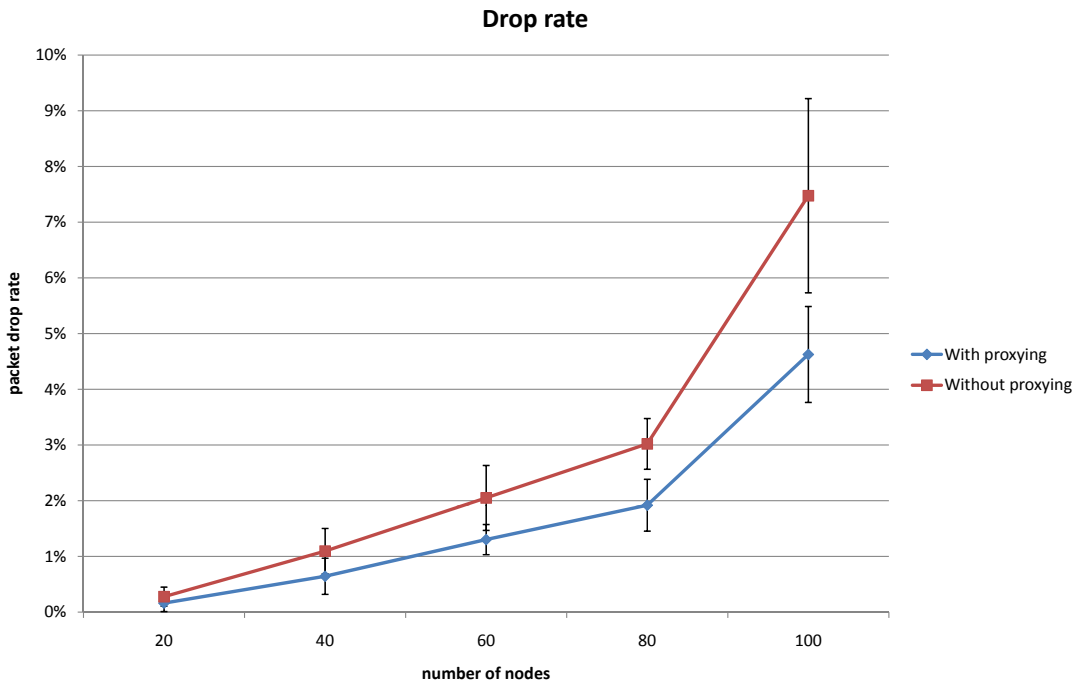


Figure 9.4: IP Packet drop rate in percent of all IP packets sent from the autoconfiguration agent (90% confidence intervals)

10 Conclusion

10.1 Summary of work

The work of this thesis consisted of different parts and different methodologies: The first part has been an analysis of current solutions in autoconfiguration and a classification of these approaches. After deficiencies of current solutions have been addressed, a new protocol specification has been outlined to circumvent these deficiencies. Several extensions and optimizations of the basic protocol have been theoretically developed. The algorithm was then formally validated with a model checker. The second part was a practical work of implementing the protocol in C++ and for the network simulator NS2. A correctness and performance analysis have been accomplished at the end of the second part.

10.2 Results

An analysis of current state-of-the-art autoconfiguration protocols for mobile ad-hoc networks (MANETs) has shown that there exist many proposals being capable of attributing unique IP addresses within a MANET. However, these protocols are all based on an architectural view which is incompatible with the current IP architecture and thus with the Internet. All proposals consider MANET nodes as hosts with IP addresses belonging to the same subnet within a MANET (and thus on the same link). This is contradictory to the properties of a MANET as a "link" in the classical sense is not applicable to MANETs where packets have to be routed over several hops in order to get from one node to another node. Within this thesis, MANET nodes are considered as *routers* with attached logical or physical hosts.

Consequently, an autoconfiguration protocol has been specified which is consistent with the architectural model of MANET nodes being routers by attributing *prefixes* and not IP addresses to MANET nodes in a first step. Next, hosts attached to these routers configure their IP addresses. In addition to that, the basic protocol proposed in this thesis is independent of any ad-hoc routing protocol or unique link-local addresses. Several optimizations have been developed that render the protocol efficient by using techniques like caching and optimized broadcasting.

A validation, simulation and real-life implementation have shown the protocol to be correct and scalable for at least hundreds of nodes. Some of the proposed optimizations have been implemented within the simulation and their performance increase has been demonstrated as substantial.

10.3 Perspective

Several issues and challenges have not been addressed in this thesis due to the limited amount of time. The issues of partitioning and merging MANETs have not been described in detail and a solution for address conflicts has still to be found. MANETs connected to the Internet have to handle global IP prefixes and to correctly manage the topology changes. Due to the mobility of the nodes, these global prefixes may have to

change when a node moves around. However, session connectivity should be assured. Another challenge for autoconfiguration is to scale the current solutions for very large MANET with tens of thousands of nodes in a MANET and millions of globally connected MANET nodes within the Internet.

Bibliography

[1] C. Adjih.
Address autoconfiguration in optimized link state routing protocol, July 2005.

[2] R. Alur and D. L. Dill.
A theory of timed automata.
Theoretical Computer Science, 126(2):183–235, 1994.

[3] IEEE Standards Association.
Group 802.11.
http://standards.ieee.org/getieee802/802.11.html.

[4] IEEE Standards Association.
Guidelines for 64-bit global identifier (EUI-64) registration authority.
http://standards.ieee.org/regauth/oui/tutorials/EUI64.html.

[5] E. Baccelli, K. Mase, S. Ruffino, and S. Singh.
Address autoconfiguration for MANET: Terminology and problem statement.
Internet Draft, work in progress, draft-baccelli-autoconf-statement-02.txt, March 2007.

[6] F. Baker.
Requirements for IP version 4 routers, June 1995.
RFC 1812, Standards Track.

[7] F. Baker.
An outsider's view of MANET.
Internet Draft, work in progress, draft-baker-manet-review-01.txt, March 2002.

[8] G. Behrmann, A. David, and K. G. Larsen.
A tutorial on uppaal, November 2004.

[9] C. Bernardos and M. Calderon.
Survey of IP address autoconfiguration mechanisms of MANETs.
Internet Draft, work in progress, draft-bernardos-manet-autoconf-survey-00.txt, July 2005.

[10] S. Boudjit, A. Laouiti, P. Muhlethaler, and C. Adjih.
Duplicate address detection and autoconfiguration in OLSR.
In *SNPD-SAWN '05: Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05)*, pages 403–410, Washington, DC, USA, 2005. IEEE Computer Society.

[11] S. Bradner.
The Internet standards process – Revision 3, October 1996.
RFC 2026.

[12] D. Cavin, Y. Sasson, and A. Schiper.
On the accuracy of MANET simulators.
In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 38–43, New York, NY, USA, 2002. ACM Press.

[13] I. Chakeres, J. Macker, and T. Clausen.
Mobile ad hoc network architecture.
Internet Draft, work in progress, draft-ietf-autoconf-manetarch-01.txt, February 2007.

[14] G. Chelius, E. Fleury, and L. Toutain.

Using OSPFv3 for IPv6 router autoconfiguration.
Internet Draft, expired, draft-chelius-autoconf-00.txt, June 2002.

[15] S. Cheshire, B. Aboba, and E. Guttman.
Dynamic configuration of IPv4 link-local addresses, May 2005.
RFC 3927, Standards Track.

[16] T. Clausen.
A MANET architectural model, January 2007.
Research Report.

[17] T. Clausen and E. Baccelli.
A simple address autoconfiguration mechanism for OLSR.
In *IEEE ISCAS '05, in proceedings*, 2005.

[18] T. Clausen, E. Baccelli, and J. Garnier.
Duplicate address detection in OLSR networks.
In *Proceedings of the IEEE Conference on Wireless Personal Multimedia Communications (WPMC)*, Aalborg, Denmark, September 2005.

[19] T. Clausen, C. Dearlove, and B. Adamson.
Jitter considerations in MANETs.
Internet Draft, work in progress, draft-clausen-manet-jitter-01.txt, February 2007.

[20] T. Clausen, C. Dearlove, and J. Dean.
MANET neighborbood discovery protocol (NHDP).
Internet Draft, work in progress, draft-ietf-manet-nhdp-02.txt, February 2007.

[21] T. Clausen, C. Dearlove, J. Dean, and C. Adjih.
Generalized MANET packet/message format.
Internet Draft, work in progress, draft-ietf-manet-packetbb04.txt, January 2007.

[22] T. Clausen, C. Dearlove, and P. Jacquet.
The optimized link-state routing protocol version 2.
Internet Draft, work in progress, draft-ietf-manet-olsrv2-03.txt, February 2007.

[23] T. Clausen, G. Hansen, and L. Christensen.
The optimized link state routing protocol, evaluation through experiments and simulation.
In *The 4th International Symposium on Wireless Personal Multimedia Comminications, in proceedings*, 2001.

[24] T. Clausen and P. Jacquet.
Optimized link state routing protocol (OLSR), October 2003.
RFC 3626.

[25] T. Clausen, P. Jacquet, and L. Viennot.
Comparative study of routing protocols for mobile ad-hoc networks.
In *In Proceeding of The First Annual Mediterranean Ad Hoc Networking Workshop*, MindPass Center for Distributed Systems, Aalborg University and Project Hipercom, INRIA Rocquencourt, September 2002.

[26] R. Coltun, D. Ferguson, and J. Moy.
OSPF for IPv6, December 1999.
RFC 2740.

[27] A. Conta and S. Deering.
Generic packet tunneling in IPv6, December 1998.
RFC 2473, Standards Track.

[28] Nokia Cooperation.
Maemo.org application development platform for Nokia Internet Tablet products.
http://maemo.org/.

[29] S. Corson and J. Macker.
Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations, January 1999.

RFC 2501, Informational.
[30] B. Croft and J. Gilmore.
Bootstrap protocol (BOOTP), September 1985.
RFC 951.
[31] A. Dimitrelis and A. Williams.
Autoconfiguration of routers using a link state routing protocol.
Internet Draft, expired, draft-dimitri-zospf-00.txt, October 2002.
[32] R. Droms.
Dynamic host configuration protocol, March 1997.
RFC 2131, Standards Track.
[33] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney.
Dynamic host configuration protocol for IPv6 (DHCPv6), July 2003.
RFC 3315, Standards Track.
[34] K. Fall and K. Varadhan.
The network simulator – NS-2.
http://www.isi.edu/nsnam/ns/.
[35] Association for Computing Machinery.
The ACM international symposium on mobile ad hoc networking and computing (MobiHoc).
http://www.sigmobile.org/mobihoc. Page accessed on April 23, 2005.
[36] B. Gebremichael, F. Vaandrager, and M. Zhang.
Analysis of the zeroconf protocol using uppaal.
In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 242–251, New York, NY, USA, 2006. ACM Press.
[37] GNU.
GCC, the GNU Compiler Collection.
http://gcc.gnu.org.
[38] B. Haberman and J. Martin.
Automatic prefix delegation protocol for internet protocol version 6 (IPv6).
Internet Draft, expired, draft-haberman-ipngwg-auto-prefix-02.txt, February 2002.
[39] U. Herberg.
Personal webpage.
http://www.lix.polytechnique.fr/herberg.
[40] R. Hinden and S. Deering.
Internet protocol version 6 IPv6 addressing architecture, April 2003.
RFC 3513, Standards Track.
[41] R. Hinden and B. Haberman.
Unique local IPv6 unicast addresses, October 2005.
RFC 4193, Standards Track.
[42] R. Hinden, M. O'Dell, and S. Deering.
An IPv6 aggregatable global unicast address format, July 1998.
RFC 2374, Standards Track.
[43] P. Hofmann.
Multihop radio access network (MRAN) protocol specification.
Internet Draft, expired, draft-hofmann-autoconf-mran-00.txt, March 2006.
[44] C. Huitema and B. Carpenter.
Deprecating site local addresses, September 2004.
RFC 3879, Standards Track.
[45] C. Jelger.
MANET local IPv6 addresses.
Internet Draft, work in progress, draft-jelger-autoconf-m1a-01.txt, October 2006.
[46] C. Jelger, T. Noel, and A. Frey.

Gateway and address autoconfiguration for IPv6 adhoc networks.
Internet Draft, work in progress, draft-jelger-manet-gateway-autoconf-v6-03.txt, November 2004.

[47] I. Jeong, H. Choi, and J. Ma.
Study on address allocation in ad-hoc networks.
In *ICIS '05: Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pages 604–609, Washington, DC, USA, 2005. IEEE Computer Society.

[48] J. Jeong, J. Park, H. Kim, H. Jeong, and D. Kim.
Ad hoc IP address autoconfiguration.
Internet Draft, work in progress, draft-jeong-adhoc-ip-addr-autoconf-06.txt, January 2006.

[49] D. Johnson, D. Maltz, and Y. Hu.
The dynamic source routing protocol.
Internet Draft, work in progress, draft-ietf-manet-dsr-10.txt.

[50] S. Kurkowski, T. Camp, and M. Colagrosso.
MANET simulation studies: the incredibles.
SIGMOBILE Mob. Comput. Commun. Rev., 9(4):50–61, 2005.

[51] 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)*, pages 503–506, 2005.

[52] Naval Research Laboratory.
Protean protocol prototyping library (protolib).
http://cs.itd.nrl.navy.mil/work/protolib/index.php.

[53] P. L'Ecuyer.
Good parameters and implementations for combined multiple recursive random number generators.
In *Operations Research*, pages 47(1):159–164, 1999.

[54] D. Lee, J. Yoo, K. Kim, and K. Kang.
IPv6 stateless address auto-configuration in mobile ad-hoc network (T-DAD) and performance evaluation.
In *PE-WASUN '05: Proceedings of the 2nd ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 271–272, New York, NY, USA, 2005. ACM Press.

[55] J. Linton.
Automatic router configuration protocol.
Internet Draft, expired, draft-linton-arcp-00.txt, March 2002.

[56] J. Linton and G. Chelius.
Zerouter protocol requirements.
Internet Draft, expired, draft-linton-zerouter-requirements-00.txt, June 2003.

[57] O. Marce and D. Galand.
Architecture for zerouter.
Internet Draft, expired, draft-marce-zerouter-archi-00.txt, December 2002.

[58] K. Mase and Y. Owada.
Gateway aggregation protocol (GAP) for mobile ad hoc networks.
Internet Draft, work in progress, draft-mase-autoconf-gap-00.txt, May 2006.

[59] P. Mase and C. Adjih.
No overhead autoconfiguration OLSR.
Internet Draft, work in progress, draft-mase-manet-autoconf-noaolsr-00.txt, February 2006.

[60] M. Mohsin and R. Prakash.
IP address assignment in a mobile ad hoc network.
In *In proceedings, MILCOM 2002*, 2002.

[61] J. Moy.
OSPF version 2, April 1998.
RFC 2328, STD 54.

[62] T. Narten, E. Nordmark, and W. Simpson.
Neighbor discovery for IP version 6 (IPv6), December 1998.
RFC 2461.

[63] P. Nicopolitidis, M. Obaidat, G. Papadimitriou, and A. Pomportsis.
Wireless Networks.
John Wiley and Sons Ltd., 2003.

[64] Y. Noisette and A. Williams.
A framework for zerouter operations.
Internet Draft, expired, draft-noisette-zerouter-frmwk-00.txt, February 2003.

[65] R. Ogier, F. Templin, and M. Lewis.
Topology dissemination based on reverse-path forwarding (TBRPF), February 2004.
RFC 3684, Experimental.

[66] I. Park, Y. Kim, and N. Kang.
Address autoconfiguration for hybrid mobile ad hoc networks.
Internet Draft, work in progress, draft-ikpark-autoconf-haa-02.txt, October 2006.

[67] C. Perkins.
Ad Hoc Networking.
Addison-Wesley, New York, 2001.

[68] C. Perkins, E. Belding-Royer, and S. Das.
Ad hoc on-demand distance vector (AODV) routing, July 2003.
Experimental RFC 3561.

[69] C. Perkins, J. Malinen, R. Wakikawa, E. Belding-Royer, and Y. Sun.
IP address autoconfiguration for ad hoc networks.
Internet Draft, work in progress, draft-ietf-manet-autoconf-01.txt, November 2001.

[70] A. Qayyum, L. Viennot, and A. Laouiti.
Multipoint relaying: An efficient technique for flooding in mobile wireless networks.
Technical Report Research Report RR-3898, INRIA, February 2000.

[71] Y. Rekhter and T. Li.
An architecture for IP address allocation with CIDR, September 1993.
RFC 1518, Standards Track.

[72] F. Ros.
UM-OLSR plugin for NS2.
http://masimum.dif.um.es.

[73] F. Ros and P. Ruiz.
Extensible MANET auto-configuration protocol (EMAP).
Internet Draft, work in progress, draft-ros-autoconf-emap-02.txt, March 2006.

[74] S. Ruffino and P. Stupar.
Automatic configuration of IPv6 addresses for MANET with multiple gateways (AMG).
Internet Draft, draft-ruffino-manet-autoconf-multigw-03.txt, June 2006.

[75] Saleem.
Address auto-configuration in mobile ad hoc networks using OLSR, August 2005.

[76] S. Thomson and T. Narten.
IPv6 stateless address autoconfiguration, December 1998.

RFC 2462.
[77] E. Tromp.
OLSR multicast forwarding plugin.
http://sourceforge.net/projects/olsr-bmf.
[78] unknown.
UPPAAL webpage.
http://www.uppaal.com.
[79] N. Vaidya.
Weak duplicate address detection in mobile ad hoc network.
In *MOBIHOC'02*, EPFL Lausanne, June 9-11 2002.
[80] R. Wakikawa, J. Malinen, C. Perkins, A. Nilsson, and A. Tuominen.
Global connectivity for IPv6 mobile ad hoc networks.
Internet Draft, work in progress, draft-wakikawa-manet-globalv6-05.txt, March 2006.
[81] K. Weniger.
PACMAN: Passive autoconfiguration for mobile ad hoc networks.
IEEE Journal on Selected Areas in Communications, 23(3):507–519, March 2005.
[82] K. Weniger and K. Mase.
PDAD-OLSR: Passive duplicate address detection for OLSR.
Internet Draft, draft-weniger-autoconf-pdad-olsr-01.txt, June 2006.
[83] K. Weniger and M. Zitterbart.
IPv6 autoconfiguration in large scale mobile ad-hoc networks.
In *In Proceedings of European Wireless 2002*, 2002.
[84] K. Weniger and M. Zitterbart.
Address autoconfiguration in mobile ad hoc networks – current approaches and future directions.
Network, IEEE, 18(4):6–11, 2004.
[85] A. White.
Zero-configuration subnet prefix allocation using UIAP.
Internet Draft, expired, draft-white-zeroconf-subnet-alloc-01.txt, October 2002.
[86] A. White and A. Williams.
Unique identifier allocation protocol.
Internet Draft, expired, draft-white-zeroconf-uiap-01.txt, October 2002.
[87] Wikipedia.
Entity.
http://en.wikipedia.org/wiki/Entity.
[88] Wikipedia.
Process.
http://en.wikipedia.org/wiki/Process.
[89] Wikipedia.
Property.
http://en.wikipedia.org/wiki/Property_(disambiguation).
[90] IETF MANET working group.
Charter of the working group.
http://www.ietf.org/html.charters/manet-charter.html.

A Appendix

A.1 Messages for the proposed autoconfiguration algorithm

In the following the messages used for the Zerouter configuration of the routers in the MANET are depicted. The packets are based on the MANET Generalized Packet/Messages format [21]. The messages are based on the ones presented in RFC 2461 [62]. For further explanation for certain fields (e.g. the preferred and valid prefix lifetime) refer to [62].

A.1.1 Variables

- `IFACE_NAME`
  Standard MANET interface. Default is "wlan0".

- `FIRST_PREFIX_PART`
  Common site prefix. Default is "fec0::".

- `NODE_PREFIX_LENGTH`
  Length of the router prefix part in semioctets (i.e. 4 bits). Default is 8.

- `MANET_PREFIX_LENGTH`
  Length of the MANET prefix part in semioctets. Default is 4.

- `PS_TIMEOUT`
  Timeout in seconds for the waiting period once a Prefix Solicitation (PS) message has been sent. Default is 1.0.

- `PS_MAX_TRIES`
  Maximum amount of tries of sending a Prefix Solicitation. Default is 2.

- `RS_TIMEOUT`
  Timeout in seconds for the waiting period once a Router Solicitation (RS) message has been sent. Default is 6.0.

- `RS_MAX_TRIES`
  Maximum amount of tries of sending a Router Solicitation. Default is 2.

- `AC_TIMEOUT`
  Timeout in seconds for the waiting period once a Autoconfiguration Confirmation (AC) message has been sent. Default is 5.0.

- `AC_MAX_TRIES`
  Maximum amount of tries of sending an AC message. Default is 2.

- `RX_PORT`
  Source port for UDP packets. May be between 49,152 and 65,535. Default is 54322.

- `TX_PORT`
  Destination port for UDP packets. May be between 49,152 and 65,535. Default is 54321.

- `TYPE_MSG_TYPE`
  6-bit long TLV type variable used to identify messages. Default is `0x01`.

- `TYPE_PREFIX_LENGTH`
  6-bit long TLV type variable used to define the prefix length. Default is `0x02`.

- `TYPE_PREFIX`
  6-bit long TLV type variable used to define the prefix. Default is `0x03`.

- `TYPE_UUID`
  6-bit long TLV type variable used to define the sending UUID. Default is `0x04`.

- `TYPE_INITIATOR`
  6-bit long TLV type variable used to define the target UUID of the initiator node. Default is `0x05`.

- `VALUE_MSG_TYPE_PS`
  8-bit long TLV value variable used to identify PS messages. Default is `0x01`.

- `VALUE_MSG_TYPE_PA`
  8-bit long TLV value variable used to identify PA messages. Default is `0x02`.

- `VALUE_MSG_TYPE_RS`
  8-bit long TLV value variable used to identify RS messages. Default is `0x03`.

- `VALUE_MSG_TYPE_RA`
  8-bit long TLV value variable used to identify RA messages. Default is `0x04`.

- `VALUE_MSG_TYPE_AC`
  8-bit long TLV value variable used to identify AC messages. Default is `0x05`.

A.1.2 Prefix Solicitation (PS) message

An exemplary message according to [21] is depicted in figure A.1. The PS message includes the following fields:

- **IP fields**:
  - **Source Address**
    Unspecified address.

  - **Destination Address**
    Broadcast address.

  - **Hop Limit**
    255

- **UDP fields:**
  - **Source Port**
    `RX_PORT`

  - **Destination Port**
    `TX_PORT`

  - **Length**
    Length of the UDP message in octets including header and data.

  - **Checksum**
    Checksum of the UDP message including header and data.

- **MANET packet fields:**
  - **Zero**
    Eight bits all set to 0. This field serves to identify that the packet starts with a packet header.

  - **Packet Semantics**
    1 octet field specifying the composition of the packet header:
    * bit 0 (pnoseqnum): set to 1 (i.e. the packet includes a sequence number).
    * bit 1 (ptlv): set to 0 (i.e. the packet header does not include a TLV block).
    * bits 2-7: are reserved, and must each be cleared ('0').

  - **Sequence Number**
    16 bit field, specifying a packet sequence number.

- **MANET message fields:**
  - **Message Type**
    Message Type is an 8 bit field, specifying the type of message. The two most significant bits are set like defined in the following:
    * bit 7 (msguser): set to 1 (i.e. reserved for private/local use).
    * bit 6 (msgprot): set to 0 (i.e. the message type is protocol independent)

  - **Message Semantics**
    Message Semantics is an 8 bit field, specifying the interpretation of the remainder of the message header:
    * bit 0 (noorig): set to 0 (i.e. then Originator Address and Message Sequence Number are included in the message header).
    * bit 1 (nohops): set to 0 (i.e. Hop Limit and Hop Count are included in the message header).
    * bit 2 (typedep): set to 0 (i.e. the message sequence number in the message is type-independent).

    - bits 3-7: are reserved and must each be cleared ('0').

  - **Message Size**
    Message Size is a 16 bit field, specifying the size of the message counted in octets.

  - **Originator address**
    Originator Address is an identifier of length equal to address-length, which serves to uniquely identify the node that originated the message.
  - **Hop limit**
    Hop limit is an 8 bit field, which contains the maximum number of logical hops a message should be further transmitted.
  - **Hop count**
    Hop count is an 8 bit field, which contains the number of logical hops a message has traveled.
  - **Message sequence number**
    Message sequence number is a 16 bit field which contains the message sequence number.
  - **TLV Total Length**
    TLV Total Length is a 16 bit field, which contains the total length (in octets) of the immediately following TLVs.

  - **TLV Type1**
    TLV Type1 is an 8 bit field, specifying the type of the TLV. The two most significant bits have the following settings:
    - bit 7 (tlvuser): set to 1 (i.e. reserved for private/local use).
    - bit 6 (tlvprot): set to 0 (i.e. TLV type is protocol independent). The following six bytes are set to `TYPE_MSG_TYPE`.

  - **TLV Semantics1**
    TLV Semantics1 is an 8 bit field specifying the interpretation of the remainder of the TLV:
    - bit 0 (extended) and bit 1 (novalue): are both set to 0 (i.e. indicating a 8 bit TLV Length1 field and an existing TLV Value1 field).
    - bit 2 (noindex) set to 1 and bit 3 (singleindex) set to 0 (i.e. not including Index Start and Index Stop fields).
    - bit 4 (multivalue) set to 0.
    - bits 5-7: are reserved and must each be cleared ('0').

  - **TLV Length1**
    TLV Length1 is an 8 bit field indicating the length of the TLV Value1 field in octets. Default value is 1.

  - **TLV Value1**
    TLV Value1 is an 8 bit field indicating the type of the message. Default value is `VALUE_MSG_TYPE_PS`.

- **TLVs:**
  - **Message Type**
    Type: `TYPE_MSG_TYPE` / Length: 1 byte / Value: `VALUE_MSG_TYPE_PS`

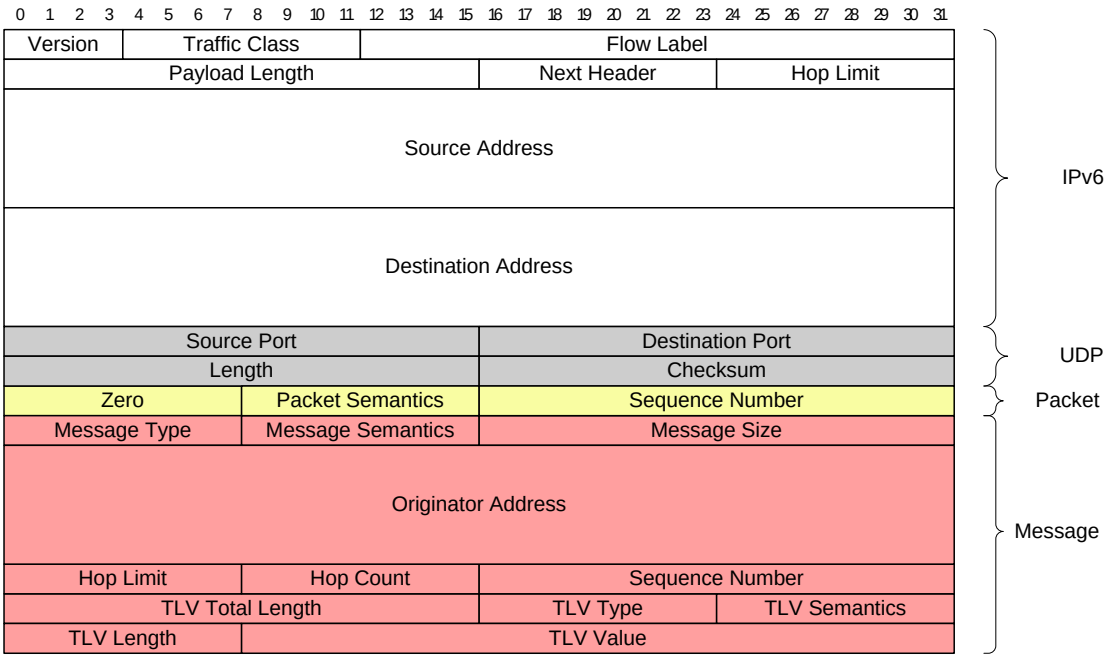


Figure A.1: Prefix Solicitation (PS) message

- **Source UUID**
  Type: `TYPE_UUID` / Length: 16 byte / Value: Source UUID of the node.

A.1.3 Prefix Advertisement (PA) message

In the following sections only the differences to the PS message of section A.1.2 are presented.

- **IP fields**:
  - **Source Address**
    The link-local IP address of the initiator node.
  - **Destination Address**
    Broadcast IP address.

- **TLVs:**
  - **Message Type**
    Type: `TYPE_MSG_TYPE` / Length: 1 byte / Value: `VALUE_MSG_TYPE_PA`
  - **Prefix Length**
    Type: `TYPE_PREFIX_LENGTH` / Length: 1 byte / Value: `VALUE_PREFIX_LENGTH`
  - **Prefix**
    Type: `TYPE_PREFIX` / Length: 16 byte / Value: `VALUE_PREFIX`
  - **Source UUID**
    Type: `TYPE_INITIATOR` / Length: 16 byte / Value: Source UUID of the node.

A.1.4 Router Solicitation (RS) message

- **IP fields**:
  - **Source Address**
    The unspecified address or the MANET IP address of the initiator node as soon as the initiator node has forwarded the RS message.
  - **Destination Address**
    Broadcast address.
- **TLVs:**
  - **Message Type**
    Type: `TYPE_MSG_TYPE` / Length: 1 byte / Value: `VALUE_MSG_TYPE_RS`
  - **Prefix Length**
    Type: `TYPE_PREFIX_LENGTH` / Length: 1 byte / Value: `VALUE_PREFIX_LENGTH`
  - **Prefix**
    Type: `TYPE_PREFIX` / Length: 16 byte / Value: `VALUE_PREFIX`
  - **Source UUID**
    Type: `TYPE_UUID` / Length: 16 byte / Value: Source UUID of the node.
  - **Target UUID**
    Type: `TYPE_INITIATOR` / Length: 16 byte / Value: Target UUID of the initiator node.

A.1.5 Router Advertisement (RA) message

- **IP fields**:
  - **Source Address**
    The MANET IP address of the conflicting or proxying node.
  - **Destination Address**
    Broadcast address or the MANET IP address of the initiator node when unicast is used.
- **TLVs:**
  - **Message Type**
    Type: `TYPE_MSG_TYPE` / Length: 1 byte / Value: `VALUE_MSG_TYPE_RA`
  - **Prefix Length**
    Type: `TYPE_PREFIX_LENGTH` / Length: 1 byte / Value: `VALUE_PREFIX_LENGTH`
  - **Prefix**
    Type: `TYPE_PREFIX` / Length: 16 byte / Value: `VALUE_PREFIX`

A.1.6 Autoconfiguration Confirmation (AC) message

- **IP fields**:
  - **Source Address**
    The link-local IP address of the initiator node.
  - **Destination Address**
    Broadcast address.

- **TLVs:**
  - **Message Type**
    Type: `TYPE_MSG_TYPE` / Length: 1 byte / Value: `VALUE_MSG_TYPE_AC`
  - **Prefix Length**
    Type: `TYPE_PREFIX_LENGTH` / Length: 1 byte / Value: `VALUE_PREFIX_LENGTH`
  - **Prefix**
    Type: `TYPE_PREFIX` / Length: 16 byte / Value: `VALUE_PREFIX`
  - **Target UUID**
    Type: `TYPE_UUID` / Length: 16 byte / Value: Target UUID of the configurating node.

A.2 Simulation scripts

The following scripts have been used for the simulation:

A.2.1 autoconf.tcl

This script launches several runs of the simulation for a given number of nodes and counts the number of conflicts.

```
#!/usr/bin/tclsh
proc getopt {argc argv} {
  global opt
  lappend optlist cp nn seed sc stop tr x y

  for {set i 0} {$i < $argc} {incr i} {
    set arg [lindex $argv $i]
    if {[string range $arg 0 0] != "-"} continue

    set name [string range $arg 1 end]
    set opt($name) [lindex $argv [expr $i+1]]
  }
}

set opt(nn) 20
set opt(run) 3
```

```
getopt $argc $argv

for {set i 1} {$i <= $opt(run)} {incr i} {
  puts "running simulation: $opt(nn) nodes  run: $i of $opt(run)"
  set scenfile "exec ./setdest -n $opt(nn) -p 1000.0 -M 3.0 \
    -t [expr $opt(nn) + 20.0] -x 1000 -y 1000 > scen-1000-$opt(nn)-$i"
  if [catch $scenfile] {
    puts "error: $::errorInfo"
  }

  set prog "exec ns autoconf-wifi.tcl -nn $opt(nn) -run $i \
     > autoconf-$opt(nn)-$i.log"
  if [catch $prog] {
    puts "error: $::errorInfo"
  }

  set fl [open "autoconf-out-$opt(nn)-$i.tr" r]
  set data [read $fl]
  set re {(?x)
        \t([[:graph:]]+)
  }

  set plist [regexp -all -inline $re $data]

  for {set j 0} {$j < [llength $plist]} {incr j} {
    set plist [lreplace $plist $j $j]
  }
  set plist [lsort $plist]

  set conflicts 0
  set oldvalue [lindex $plist 0]

  for {set j 1} {$j < [llength $plist]} {incr j} {
    set curvalue [lindex $plist $j]
    if  {$curvalue == $oldvalue} {
      incr conflicts
    } else {
      set oldvalue $curvalue
    }
  }

  puts "conflicts: $conflicts"
}
```

A.2.2 autoconf-wifi.tcl

```
# ======================================================================
# Default Script Options
# ======================================================================

set opt(chan)     Channel/WirelessChannel
```

```
set opt(prop)     Propagation/TwoRayGround
set opt(netif)    Phy/WirelessPhy
set opt(mac)      Mac/802_11
set opt(ifq)      Queue/DropTail/PriQueue
set opt(ll)       LL
set opt(ant)      Antenna/OmniAntenna

set opt(x)        1000       ;# X dimension of the topography
set opt(y)        1000       ;# Y dimension of the topography

set opt(ifqlen)   50         ;# max packet in ifq
set opt(nn)       60         ;# number of nodes
set opt(seed)     12345
#set opt(rp)      DumbAgent  ;# routing protocol script
set opt(rp)       OLSR       ;# routing protocol script
set opt(lm)       "OFF"      ;# log movement
set opt(run)      1



# ==================================================================

LL set mindelay_   50us
LL set delay_      25us
LL set bandwidth_  0  ;# not used

Agent/Null set sport_    0
Agent/Null set dport_    0


Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node
# and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10  ;# = 250 m
Phy/WirelessPhy set Rb_ 2*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0

# ==================================================================
```

```
proc usage { argv0 }  {
  puts "Usage: $argv0"
  puts "\tmandatory arguments:"
  puts "\t\t\[-x MAXX\] \[-y MAXY\]"
  puts "\toptional arguments:"
  puts "\t\t\[-cp conn pattern\] \[-sc scenario\] \[-nn nodes\]"
  puts "\t\t\[-seed seed\] \[-stop sec\] \[-tr tracefile\]\n"
}


proc getopt {argc argv} {
  global opt
  lappend optlist cp nn seed sc stop tr x y

  for {set i 0} {$i < $argc} {incr i} {
    set arg [lindex $argv $i]
    if {[string range $arg 0 0] != "-"} continue

    set name [string range $arg 1 end]
    set opt($name) [lindex $argv [expr $i+1]]
  }
}

proc log-movement {} {
    global logtimer ns_ ns

    set ns $ns_
    source ../mobility/timer.tcl
    Class LogTimer -superclass Timer
    LogTimer instproc timeout {} {
        global opt node_;
        for {set i 0} {$i < $opt(nn)} {incr i} {
            $node_($i) log-movement
        }
        $self sched 0.1
    }

    set logtimer [new LogTimer]
    $logtimer sched 0.1
}


proc create-god { nodes } {
  global ns_ god_ tracefd

  set god_ [new God]
  $god_ num_nodes $nodes
}

# ======================================================================
# Main Program
```

```
# ======================================================================
getopt $argc $argv

if {$opt(run) < 1} {
   set opt(run) 1
}

if { $opt(x) == 0 || $opt(y) == 0 } {
  usage $argv0
  exit 1
}

if {$opt(seed) > 0} {
  puts "Seeding Random number generator with $opt(seed)\n"
  ns-random $opt(seed)
}

set opt(cp)         "./autoconf-1"
set opt(sc)         "./scen-$opt(x)-$opt(nn)-$opt(run)"
set opt(tr)         autoconf-out-$opt(nn)-$opt(run).tr ;# trace file
set opt(stop)       [expr $opt(nn) + 20.0]  ;# simulation time

#
# Initialize Global Variables
#
set ns_             [new Simulator]
set chan            [new $opt(chan)]
set prop            [new $opt(prop)]
set topo            [new Topography]
set tracefd         [open $opt(tr) w]

#set nf [open autoconf-$opt(nn)-$opt(run).nam w]
set f [open autoconf-$opt(nn)-$opt(run).tr w]

#$ns_ namtrace-all-wireless $nf $opt(x) $opt(y)
$ns_ trace-all $f
$ns_ use-newtrace

$topo load_flatgrid $opt(x) $opt(y)

$prop topography $topo

#
# Create God
#
create-god $opt(nn)
log-movement

#
#  Create the specified number of nodes $opt(nn) and "attach" them
#  the channel.
#  Each routing protocol script is expected to have defined a proc
```

```
#  create-mobile-node that builds a mobile node and
#  inserts it into the array global $node_($i)

$ns_ node-config \
                 -adhocRouting $opt(rp) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propInstance $prop \
                 -phyType $opt(netif) \
                 -channel $chan \
                 -topoInstance $topo \
                 -wiredRouting OFF \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace $opt(lm)


#enable node trace in nam

for {set i 0} {$i < $opt(nn)} {incr i} {
    set node_($i) [ $ns_ node ]
    $node_($i) random-motion 0
    $ns_ initial_node_pos $node_($i) 20
}


#
# Source the Connection and Movement scripts
#
if { $opt(cp) == "" } {
  puts "*** NOTE: no connection pattern specified."
       set opt(cp) "none"
} else {
  puts "Loading connection pattern..."
  source $opt(cp)
}


#
# Tell all the nodes when the simulation ends
#
for {set i 0 } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i) reset";
}
$ns_ at $opt(stop).1 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop) "stop"

if { $opt(sc) == "" } {
```

```
  puts "*** NOTE: no scenario file specified."
       set opt(sc) "none"
} else {
  puts "Loading scenario file..."
  source $opt(sc)
  puts "Load complete..."
}

puts "Starting Simulation..."
$ns_ run

proc stop {} {
    global ns_ f nf
    $ns_ flush-trace
    close $f
    close $nf
}
```

A.3 Routing agent changes

A.3.1 OLSR.h

```
359d358
<     bool is_activated;  // is forwarding enabled?
```

A.3.2 OLSR.cc

```
39,40d38
< #include <iostream>
<
100d97
<  is_activated = true;
104,109d100
<  // returns node address
<  else if(strncasecmp(argv[1], "id", 2) == 0) {
<    Tcl& tcl = Tcl::instance();
<    tcl.resultf("%d", ra_addr_);
<    return TCL_OK;
<  }
248,261d238
<  else if (argc == 5){
<    if (strcmp(argv[1], "add-route") == 0) {
<      int dst = atoi(argv[2]);
<      int nexthop = atoi(argv[3]);
<      int ownAddr = this->addr();
<      int dist = atoi(argv[4]);
<
<      if (rtable_.lookup(dst) == NULL){
<        rtable_.add_entry(dst, nexthop, ownAddr, dist);
<      }
<    }
<    return TCL_OK;
```

```
<  }
467d443
<  is_activated = false;
484c460
<
---
>
495d470
<
528c503
<  if (!is_activated ||
       op->pkt_len() < OLSR_PKT_HDR_SIZE + OLSR_MSG_HDR_SIZE) {
---
>  if (op->pkt_len() < OLSR_PKT_HDR_SIZE + OLSR_MSG_HDR_SIZE) {
1199d1173
<
1256,1258d1229
<     if (!is_activated)
<    return;
<
1354,1356d1324
<     if (!is_activated)
<    return;
<
1389,1391d1356
<     if (!is_activated)
<    return;
```